Applied Ontology 20 (2012) 1–3
IOS Press

A Type-Theoretical Approach for Ontologies: the Case of Roles.

Patrick Barlatier [a], Richard Dapoigny [a,*]
[a] *LISTIC/Polytech'Annecy-Chambéry*
University of Savoie, Po. Box 80439, 74944 Annecy-le-vieux cedex, France
E-mail: {patrick.barlatier,richard.dapoigny}@univ-savoie.fr

Abstract. In the domain of ontology design as well as in Knowledge Representation, modeling universals is a challenging problem. Most approaches that have addressed this problem rely on Description Logics (DLs) but many difficulties remain, due to under-constrained representation which reduces the inferences that can be drawn and further causes problems in expressiveness. In mathematical logic and program checking, type theories have proved to be appealing but, so far they have not been applied in the formalization of ontologies. To bridge this gap, we present in this paper a theory for representing ontologies in a dependently-typed framework which relies on strong formal foundations including both a constructive logic and a functional type system. The language of this theory defines in a precise way what ontological primitives such as classes, relations, properties, etc., and thereof roles, are. The first part of the paper details how these primitives are defined and used within the theory. In a second part, we focus on the formalization of the role primitive. A review of significant role properties leads to the specification of a role profile and most of the remaining work details through numerous examples, how the proposed theory is able to fully satisfy this profile. It is demonstrated that dependent types can model several non-trivial aspects of roles including a formal solution for generalization hierarchies, identity criteria for roles and other contributions. A discussion is given on how the theory is able to cope with many of the constraints inherent in a good role representation.

Keywords: Ontology, Dependent Types, Categories, Relation Type, Role Type, Context, Subsumption, Generalization hierarchies

1. Introduction

Despite the increasing interest in the role of foundational ontologies for conceptual modeling and knowledge representation, most modeling languages that have been proposed so far to express ontological constraints (or rules) are based on a very simple meta-conceptualization as underlined in Guizzardi-et-al (2006). Usual languages offer appropriate structuring mechanisms such as classes, relationships and subsumption (subclass relations). However, in the representation of a formula, some structures have meaning whereas others do not make sense. This aspect requires "suitable ontological distinctions" understood as meta-properties of ontological structures as pointed out in Guarino (1994). The principle of identity or rigidity are such meta-properties. In addition, the distinction of ontological meta-level categories such as types, kinds, roles, relations, etc., further make accurate and explicit the real-world semantics of the terms that are involved in domain representations. Not only is an ontology committed to represent knowledge of reality in a way that is independent of the different uses one can make of it, but it is intended to provide a certified and coherent map of a domain. All these constraints can only be fulfilled within a highly expressive language built on a solid logical background. For that purpose in this paper, we propose a two-layered theory including a higher-order dependent type theory as lower layer and an ontological layer as upper layer. While type theory is widely used in mathematical logic Barendregt (1997, 2001); Martin-Lof (1982); Andrews (1986) and in Computer Science Barendregt (1992); Constable (1988); Reus (1999); Oury (2008); Hancock (2000); Luo (1994), there are few works using it for knowledge representation. The proposed theory called K-DTT (Knowledge-based Dependent Type Theory) is derived from Dapoigny (2010b) for the modeling of contexts and from Dapoigny (2012) for the modeling of situations. The logic in the lower layer operates on (names of) types whose meaning is constrained in the upper (ontological) layer.

*Corresponding author: Richard Dapoigny.

The core ingredients of dependent type theory are universes and dependent types. Type theory is stratified in a cumulative way with universes (types of types) in order to solve the inconsistency: $Type$ is of type $Type$. Adopting a stratified hierarchy of universes results in a very powerful system, in which every universe includes the option of talking about the types nested in the universe below. Dependent types are types indexed by some value and give rise to indexed families of types. These indexed families provide a high expressiveness since they can represent subset types, relations or constraints as typed structures on which reasoning is achieved in the form of sequences transforming each type in another one at each step. The dependency could be used to enforce semantic conditions as we will see in section 4. Dependent types and universes will be exploited for representing knowledge in an elegant and secure way. The last aspect is analyzed in Cirstea-et-al (2004) where the authors investigate typing applied to reasoning languages of the Semantic Web and point out that dependent types ensure normalization. For example, type theory enjoys the property of subject reduction which expresses that no illegal term will appear during the execution of a well-typed query in a well-typed program. By contrast, languages using some kind of untyped logic (e.g., FOL) allow to assert axioms which are not well-formed. In addition, many foundational ontologies have lexical description and axioms in some controlled language as completely separated components. From an engineering perspective, it has been proved advantageous in a recent work of the FP7 European project Enache (2011), to use a common language for both. Furthermore, the authors have shown the ability of the type-theoretical approach to cope with scalability on the SUMO foundational ontology.

Type theory is a generalization of higher-order logic with the corresponding expressive power and has variables of many different sorts imposing significant constraints on the ways in which these variables can be combined. However, a general objection against higher-order logic is its computational intractability. This argument is no longer valid if we consider a higher-order dependent type theory such as the Calculus of Constructions (CC) Coquand (1988) and its extended version, the Extended Calculus of Constructions (ECC) Luo (1994) since their computational property stems from the Curry-Howard isomorphism (more details will be given in section 2.2). The remaining difficulty is that most ontologists and researchers in the area of AI seem unaware of the benefits that a type theory can offer for knowledge representation.

On the one hand, the topic of type theory is fundamental both in logic and computer science Reynolds (1983, 1984) while on the other hand the topic of ontology[1] is challenging from the perspective of knowledge representation and reasoning. In order to bridge the gap between these two domains, we propose the K-DTT approach extending the works in Dapoigny (2010a,b). We introduce a foundational approach departing from the existing ones such as usual first-order logic and set-based theories with the purpose of providing a simple, coherent and tractable theory. We will demonstrate that the proposed theory is able to satisfy most of all the constraints inherent in an expressive knowledge representation. To reach this objective, we discuss the impact of an expressive and rigorous ontological language in a first part and then, address the particular problem of role modeling Guarino (1992); Steimann (2000); Masolo-et-al (2004); Welty (2001); Guizzardi (2005)) in a second part. In Section 2, we present in an informal way the major characteristics of the type-theoretical framework. In Section 3, we summarize the basic syntax of the type theoretical layer of K-DTT. Section 4 outlines the terms of the ontological layer language for representing high-level notions such as classes, relations, properties and subsumption. Section 5 enumerates a small number of properties that a role structure should satisfy from a review of related works while Section 6 explains how the representation of roles can be captured in K-DTT. Section 7 focuses on the mechanism of subsumption and its ability to address formally the generalization hierarchies problem. Section 8 considers some role properties such as the identity problem with roles, roles of roles and solves in a formal way the so-called counting problem. The results obtained in the proposed theory are discussed and followed by the conclusion in Section 9.

[1] Understood here as ontology in computer science.

2. Motivations for a Type-Theoretical Framework

2.1. Towards a Type-Theoretical Framework

From the perspective of facilitating large-scale knowledge integration, there is a need to constrain the language primitives of conceptual models. These primitives have a (subjective) cognitive interpretation and for that purpose, require an ontological approach in order to restrict their semantics, as advocated e.g., in Guizzardi (2004). The term ontology here refers to a formal ontology in Computer Science understood as the general theory of being. A formal ontology is precisely concerned with semantics, the actual meaning of different notions or beings and deals with concepts such as whole, part, event, process, etc. Alternatively, logic defines predicates and functions establishing relations between objects of the domain without any assumption on the nature of these relations. Assuming a layered structure including a logical level subsumed by an ontological level, the meaning of basic ontological categories in the domain (i.e., the number of possible interpretations) is restricted in the ontological level.

One has to be aware of the dual nature of a dependent type theory (the so-called Curry-Howard isomorphism) leading to consider the theory both as a (constructive) logic and a type system Coquand (1988); Luo (1994). As a consequence, we have the benefit to exploit a logic that is neutral from an ontological perspective together with (names of) types whose meaning will be constrained in the ontological level. This last aspect is particularly appealing for restricting the terms of the language. All these aspects are embedded in K-DTT which extends the underlying type theory with structures expressed by appropriate data types giving rise to what is known as a computational theory Luo (1994). It relies on a dependent type theory, the proof-irrelevant Extended Calculus of Constructions Luo (1994); Werner (2008) which has been complemented at the ontological level (more details about this aspect are given in the next section) with some features well suited for Knowledge Representation. As a consequence, all the ontological classes introduced in the following must have corresponding structures satisfying typing mechanisms provided at the logical level. In summary, the basic idea of the present theory is to supplement the so-called Curry-Howard isomorphism with ontological classes.

We will now explain in an informal way how the present type-theoretical framework: (i) exploits the Curry-Howard isomorphism to replace natural deduction with typing reductions and relates typing to ontological categories, (ii) is able to express basic modal assertions, (iii) adopts an intermediary position between the closed-world and open-world assumptions in order to remain consistent with both knowledge bases and data bases and (iv) uses stratified type universes to formalize e.g., meta-properties seen as properties of properties with distinct levels of representation as in a metalanguage.

2.2. Using the Curry-Howard Isomorphism

Type theory is known to be both a type system Whitehead (1997) and a constructive logic. Let us clarify these two central paradigms.

First, let us recall that a typed λ-calculus has been shown to be appropriate for representing knowledge and more especially, ontological knowledge Uschold (1996). Among the numerous versions about the description of what a concept is, the Fregean perspective considers a concept as a function whose input value is an object and whose output value is always a truth value. Concepts are proved if and only if the object falls under the concept. For example the meaning of "John drinks" can be represented by a function argument expression $A(x)$ where A denotes a function ("drinks") and x an argument to that function with a possible value a which stands for "John". It is well known that one way to make this syntax computational is by means of lambda notation. The function expression of this example is then written as the usual β-reduction[2] $(\lambda x\ .\ A)a = A[a/x]$, in which $A[a/x]$ is the term that results from A by substituting term a for the free occurrences of the variable x in A. Therefore, it is worth considering concepts as the result of

[2]The process of substituting a bound variable in the body of a lambda-abstraction by the argument passed to the function whenever it is applied.

a lambda calculus. We follow that line for concepts and relations and argue that the theoretical framework should be supported by a functional language because it natively supports the primitives of a lambda-calculus (e.g., Lisp, ML, etc.). While the idea of using a lambda calculus is of interest for representing knowledge, it is not sufficiently expressive and must be complemented with a typing mechanism. The idea of typing, which first appears in the *Principia Mathematica* of Whitehead (1997), has been followed by a typed version of the lambda calculus proposed in Church's type theory, a formal logical language including First-Order Logic (FOL), but which is more expressive in a practical sense. Many typed versions have been proposed so far, such as the polymorphic second-order typed lambda calculus $\lambda 2$ known as "system F" Girard (1971) or λP with types depending on terms (see the AUTOMATH project de Bruijn (1987)). These theories have been followed by more expressive systems known as higher-order polymorphic Dependent Type Theories, e.g., the Calculus of Constructions Coquand (1988) and the Extended Calculus of Constructions Luo (1994) including more powerful features (e.g., cumulativity and/or sum-types).

Second, a type theory is not restricted to support knowledge representation but also provides a (constructive) higher-order logic able to reason about knowledge structures. In the classical (denotational) Tarskian approach, any proposition is either true or false. Then, the truth of a formula results from a computation of all truth values for each term that appears in this formula. Conversely, the underlying logic of most type theories is constructive in the Heyting sense[3]. The Heyting view considers that the truth of a formula is proved iff we have a means (function or constructor) able to prove it. Furthermore, it has been shown that a typed λ-calculus is "equivalent" to a proof system in intuitionistic (i.e., constructive) logic according to the so-called Curry-Howard isomorphism Howard (1980). It states a correspondence between (i) a formalism for expressing effective functions (the λ-calculus) and (ii) a formalism for expressing proofs (natural deduction for intuitionistic logic). Type theories make use of this isomorphism viewing propositions as types while proofs are seen as objects. In other words they establish an "equivalence" between a computation system (i.e., a lambda-calculus) and a logical theory. They also have explicit proof objects which are terms in an extension of the typed lambda-calculus while at the same time, provability corresponds to type inhabitation[4]. Proofs in a computational setting can be the result of a database lookup, the existence of a function performing a given action or the output of a theorem prover, given assumptions about entities, properties or constraints. If we exhibit a proof for a proposition, then the proposition is proved and it follows that the proposition is said to be true. However, a proof is built up of premises, and using different environments, the same assertion can have different meanings. As a result, a proposition is identified with the set of its proofs rather than verifying worlds like in FOL.

Since there is an equivalence between a logic and a typing mechanism, the use of type theory replaces logical derivations with (computational) typing reductions, i.e., typing is proving. While type theories have been extensively used in program verification and mathematics, in the area of knowledge representation, they have been mainly used in Natural Language Processing Montague (1970); Muskens (1996); Cimiano (2003); Asher (2008), and in a more expressive way with Ranta (2004); Boldini (2000); Cooper (2005). The limitation in the development of these theories is first the lack of researchers having both type-theoretical and ontology-based knowledge and second, the lack of tools allowing non familiar users of type theory to take advantage of its benefits. In the present work, it is to address the first limitation that we (i) propose a type-theoretical framework for representing knowledge under ontological assumptions (ii) use this theory to address in a unified framework current problems in role representation and (iii) test the theoretical assumptions with a well-known theorem prover.

To reach these objectives, type theory is used as a uniform computational language for programming, specification and reasoning where basic ontological structures are introduced in the language as data types. In such a way, the type theory can be used as a programming language due to its computational features. Furthermore, the rich type structure together with the higher-order internal logic allow one to consider (meta) reasoning about structured ontological specifications.

[3] Also called the Brouwer-Heyting-Kolmogorov (BHK) semantics.

[4] A proposition is true iff its set of proofs is non empty.

2.3. Expressing Modality

Whether K-DTT addresses modal capabilities is a crucial question which must be clarified. These capabilities appear first in modal logic which extends classical predicate logic to include operators expressing modality in order to qualify the truth of a judgment. There is a family of modal logics (e.g., K, $S4$, $S5$) constructed from basic modal operators including "necessarily" as in $\Box\phi$ (read as "it is necessary that ϕ") and "possibly" as in $\Diamond\phi$ ("It is possible that ϕ"). Furthermore, to prove both soundness and completeness of a modal logic, a formal semantics is required to define its validity and to characterize the truth behavior for all the sentences of its related language. The standard semantics for modal logics is the "possible worlds" semantics also known as Kripke semantics Kripke (1963). A valuation ascribes a truth value to each propositional variable for each of the possible worlds in a set of possible worlds. A possible statement in modal logic is said to be true in at least one possible world while a necessary statement is said to be true in all possible worlds. There is a close relation between propositions and possible worlds since every proposition is either true or false in any given possible world. In the language of possible-world semantics, a statement is (i) logically indeterminate (contingent) if and only if it is true in some possible worlds and false in others, (ii) logically true if and only if it is true in all possible worlds and (iii) logically false (contradictory) if and only if it is false in all possible worlds.

The possible worlds semantics is implicitly taken into account in the type-theoretical layer of K-DTT when considering the precise meaning for the terms truth, proposition and hypothetical judgment.

Truth Since K-DTT relies on Intuitionistic Logic (IL), it follows that the fundamental (intuitionistic) notion of truth is formulated as 'the propositional content P is true if and only if P is proved' and P is proved if we can construct a proof for it. In other words, IL relies on a strengthening of the concept of truth, by translating truth-conditions into proof-conditions, and hence can be seen as a logic of explicit justification. Notice that if a type is not proved in IL that does not mean that it is false (it corresponds to contingent statements of possible world semantics). We would accept P in IL if P is provable, therefore if P is provable, it should be true: $Provable(P) \supset P$. By using the standard interpretation of the modal operator $\Box$ a basic axiom of the modal logic $S4$ is: $\Box P \supset P$ (axiom (T)). Since we expect to prove only formulas that are actually true, then provability is identified with $\Box$.

Proposition In possible world semantics a proposition is viewed as the set of possible worlds in which it is true whereas in K-DTT a proposition is regarded as a type whose last component is of type $Prop$[5] (e.g., $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow Prop$). This type is instantiated by a proof (here the proof is unique) that it is true. Here, proofs must be constructed for every proposition and this is a major difference with possible world semantics.

Hypothetical judgment Broadly speaking, a judgment is something that is knowable. The type-theoretical layer of K-DTT does not only express reasons for knowledge contents as terms in its language, but it distinguishes between *categorical judgments* and *hypothetical judgments* by means of syntactically different justification terms in the language. Each typing judgment is a truth assertion $p : P$ (read "p is of type P") for the propositional content "P is true" with $P : Prop$. In the lower layer of K-DTT, an environment is composed of a sequence of variables $(x_1, x_2, \ldots, x_n)$, each term being assigned to a variable. *Categorical judgments* represent a propositional content ascribed to proofs in the empty environment, i.e., they require no assumption for their validity (non-dependent case). Alternatively, *hypothetical judgments* assert that a judgment $p : P$ depends on hypotheses or assumptions $(x_1, x_2, \ldots, x_n)$. The semantic content of these hypotheses reflects in the upper layer the content of an ontology such as concepts, relations and possibly the content of a database, while the semantics of P may denote rules such as the transitivity of part-of relations. The use of dependent justifications expresses potential (or implicit) knowledge in the sense where (i) all the needed conditions $(x_1, x_2, \ldots, x_n)$ make the truth of P assertable and (ii) p is true if the truth of P has been proved, that is a construction p of P is known (constructive principle). The former is a necessary

[5]The type $Prop$ is the type of propositions.

but not a sufficient condition. It follows that if the environment is valid (there exists a proof for each variable), we get the potential knowledge that there is an assertion $p : P$. It is only when a construction p of P is known that the judgment becomes valid (proved).

An explicit way to give an account of the logical structure of possibility and necessity talk is to observe that IL gives us a possible-world semantics for free, thanks to the embedding of IL into $S4$, proved in Gödel (1933). We refer here to the Heyting's version Heyting (1930) of Brouwer's intuitionistic propositional logic in terms of the modality 'it is provable' (see above). However, modal logic, inside which intuitionistic connectives can be faithfully translated, is non-constructive. It has been shown that the various logics which have been considered so far present themselves in the best case as "impoverishments of intuitionism" Girard (2006). Therefore, the intuitionistic basis of the K-DTT lower layer appears sufficiently expressive to take in account the modal aspects inherent in ontological structures.

2.4. Supporting the Regular World Assumption

On the one hand, databases describe a closed world, that is, only those entities together with their related properties which are represented in the system exist. Any facts that have unknown truth values cannot be represented and the closed-world assumption assumes that anything which is not contained within the data base is assumed to be false (i.e., unknown is equivalent to false). On the other hand, ontologies rather support the open-world assumption and related languages such as Description Logics also face such a difficulty. So, the question arises of what can be the impact of the closed world assumption in type theory. First, the closed world assumption in type theory would require that arguments of functions must be closed, i.e., functions could not be defined on free parameters. Second, it is well known in logic that open world assumption is typical of languages with embedded implication and universal quantification. Third, the universe construction renders the theory strictly open-world. Fourth, from the perspective of constructive logic, the domain is open since new proof objects can be added at each occurrence of a type checking on the knowledge base. For predicates (i.e., types in $Prop$), unknown facts are not proved and this does not imply that they are false (this is a property of IL). As stated before, adding a database for proving types in K-DTT requires the closed-world assumption for type-inhabitation on the database (at a given time). With regard to these constraints, the adopted solution is to restrict the set of proof objects we deem inhabited so as to enforce a Regular Word Assumption (RWA) Schürmann (2001). For that purpose, we define a class of proof objects whose dynamic assumptions extend the current database in a specific regular way. It results that the set of proof objects in the database is a subset of the possible proof objects for K-DTT.

2.5. Using a Hierarchical Structure

The type-theoretical language should represent and reason (with a higher-order logic that is internal to the language) about knowledge structures in a formal way with a type hierarchy. The idea of a type hierarchy originates with Russell in the *Principia Mathematica* which thought it necessary to introduce a kind of hierarchy, the so-called "ramified hierarchy"Russell (1986). One should make a distinction between the first-order properties, that do not refer to the totality of properties and consider that the second-order properties refer only to the totality of first-order properties. This clearly eliminates all circularities connected to impredicative[6] definitions (e.g., $Type : Type$). While any term has a type in a type theoretical framework, one can wonder if a type can itself have a type. The answer is yes, and leads naturally to what is called a universe (a collection of types). These universes can be partially ordered and are organized into a (non finite) hierarchy of universes called the universe hierarchy. The universe hierarchy may rely on the notion of impredicativity because impredicative systems integrate an important idea of polymorphic type which allows quantification over all propositions or types to form a new proposition or type. For example, the output of $\forall P : Prop.P$ is a proposition (of type $Prop$) which is impredicative since it is formed by

[6]Impredicativity is a kind of conceptual circularity.

quantifying over all propositions including itself. The use of impredicative universes gives the system both a strong logical power Barendregt (1992) (i.e., the expressiveness of the intuitionistic higher-order logic) and a strong computational power (see Girard (1973)). If we assume that both universes for types and the universe for propositions are impredicative, then using propositions to represent data types would result in a non-conservative extension of higher-order intuitionistic logic (see Luo (1990) for more details). Following Luo (1992), a possible solution to this problem is to state that there is a formal distinction between the notions of logical formula and data type in the type-theoretical layer. It yields a conceptual universe of types which consists in two parts, an impredicative universe $Prop$ for propositions and a predicative hierarchy of data types (in the spirit of Russell). The resulting type theory Luo (1992, 1994); Werner (2008) at the basis of the lower layer in K-DTT is (globally)[7] impredicative.

The theory comprises an infinite hierarchy of predicative type universes denoted $Type_0$, $Type_1$, ... and an impredicative universe noted $Prop$. The universes $Type_i$ are the universes for data types while $Prop$ is the universe for logic. The hierarchy is cumulative, that is, $Type_i$ is contained in $Type_{i+1}$ for every i. A universe is seen as a type that is closed under the type-forming operations of the calculus. Since impredicativity exists only for $Prop$[8], $Type_i$ may contain only types from universes $Prop$, $Type_0$, ..., $Type_{i-1}$ while $Prop$ may be constructed with types from $Prop$, $Type_0$, ..., $Type_i$, etc. Viewing types as sets, we have intuitively $Prop \subseteq Type_0 \subseteq Type_1 \subseteq ....$ The hierarchy of universes is partially ordered by a cumulativity relation $\preccurlyeq$ also called subtyping in the following part of the paper.

Lemma 1. *Luo (1994) The cumulativity relation $\preccurlyeq$ is the smallest partial order over terms such that $Prop \preccurlyeq Type_0 \preccurlyeq Type_1 \preccurlyeq ... \preccurlyeq Type_i$....*

where $i \in \mathbb{N}$ and $Type_i$ are type universes. Since a term within a universe may quantify over terms in the lower universe, stratification has a natural capability for representing meta-reasoning.

3. K-DTT: the Type-Theoretical Layer

3.1. Syntactic Sugar

The terms of the Type-Theoretical layer are generated by the following grammar Luo (1994):

$$\mathcal{T} ::= \mathcal{V} \mid Prop \mid Type_i \mid \Pi\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \lambda\mathcal{V} : \mathcal{T}.\mathcal{T} \mid (\mathcal{T}\mathcal{T}) \mid \Sigma\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \langle\mathcal{T}, \mathcal{T}\rangle_{\mathcal{T}} \mid \pi_1(\mathcal{T}) \mid \pi_2(\mathcal{T})$$

in which, $\mathcal{V}$ is a set of variables, $Prop$ and $Type_i$ ($i \geqslant 0$), are type universes as described above, $\langle\mathcal{T}, \mathcal{T}\rangle_{\mathcal{T}}$ stands for pairs in which the subscript denotes the type of the pair. A pair of elements M and N is written as $\langle M, N\rangle_T$, the type T of the resulting pair has to be added for reasons of type inference, but when no confusion occurs we will abbreviate $\langle M, N\rangle_T$ as $\langle M, N\rangle$. In what follows, we denote by Γ an environment to avoid confusions with the notion of context. Each typing assertion is made with first checking its environment. The environment is a finite sequence[9] of expressions of the form $x_i : T_i$ where x_i is a variable and T_i a term. The fundamental notion of typing judgment $\Gamma \vdash M : T$ where M and T are terms is read as "M has type T in Γ".

Definition 1. *Let Γ be a valid environment.*
A term T is called a type in Γ if $\Gamma \vdash T : U$ for some universe U.
A term M is called an object in Γ if $\Gamma \vdash M : T$ for some type T.

A concise system of rules govern the behavior of the type-theoretical layer. They can be broadly divided into three categories, type formation rules, introduction rules and elimination rules. Type formation rules define how elements of the type universes $Prop$ and $Type_i$ can be constructed. Introduction rules, describing how canonical elements of the respective dependent types are formed. The elimination rules define how elements of dependent types can be applied.

[7] Since a part of the system is impredicative, the whole system remains impredicative.
[8] $Prop$ is seen as an object of $Type_0$.
[9] The sequence is ordered because any type in the sequence may depend on the previous variables.

Definition 2. *A term M is well-typed if for some environment Γ we have $\Gamma \vdash M : T$ for some T.*

Definition 3. *A type T is inhabited in the environment Γ if $\Gamma \vdash M : T$ for some M.*

To show that $M : T$ holds in a given environment Γ, one has to show that either Γ contains that expression or that it can be obtained from the expressions in Γ with the help of type deduction rules. Notice that a term has a single type (up to subtyping) while a proof can be related to several terms.

While types can be as simple as e.g., the type "int" which stands for integers, more interesting types are dependent types, i.e., dependent sums (also called Σ-types or sum types) and dependent products (also referred to as Π-types or product types). The characteristic feature of dependent type theories is precisely that they allow a type to be predicated on a value. This property makes them much more flexible and expressive than conventional type systems McKinna (2006). A core notion is that a family of types is indexed by a term (having itself a type). Product types are functions whose result type depends on the argument while sum types denote pairs, where the first element determines the type of the second element. The non-dependent versions of these types are respectively denoted by $\rightarrow$ (functions) and by $\times$ (non-dependent pair).

The product type has functions as objects. The intuitive map from x to $M[x]$ is written $\lambda x\ .\ M[x]$. For a type A and any family of types $B[x]$ indexed by arbitrary objects x of type A,

$$\Pi x : A\ .\ B[x]$$

is the type of functions f such that for any object a of type A, applying f to a yields an object of type $B[a]$. Intuitively it represents the set of (dependent) functions from A to $B[x]$:

$$\{f \mid fa\ :\ B[a],\ \forall a : A\}$$

Thus $B[M/x]$ is a type whenever M is a term of type A. As a consequence, product types express the universal quantification $\forall$.

Sum types are types of pairs of objects. For any type A and any family of types $B[x]$ indexed by arbitrary objects x of type A,

$$\Sigma x : A\ .\ B[x]$$

is the type of pairs $\langle a, b\rangle$ where a is an object of type A and b is of type $B[a]$. Intuitively it represents the set of (dependent) pairs of elements of A and $B[x]$:

$$\{\langle a, b\rangle \mid a\ :\ A,\ b\ :\ B[a]\}$$

When B is a predicate over A, it expresses the subset of all objects of type A satisfying the predicate B[10]. In each pair s, π-elimination rules extract the first and the second component with the respective functions $\pi_1(s)$ and $\pi_2(s)$. In the following part of the paper we drop the parentheses for the sake of simplicity. As a consequence, we can express a subset with a dependent sum type: $\Sigma x : S\ .\ P[x]$ which corresponds to the set-based expression: $\{x \in S \mid P[x]\}$.

The type forming rules for dependent types, and more precisely for sum types allow to compute the resulting universe of a dependent type as follows.

Lemma 2. *Luo (1994); Coq Development Team (2010) Let A, B, two types defined in the current environment Γ such that $\Gamma \vdash s : (\Sigma x : A\ .\ B[x])$, then the universe of $\Sigma x : A\ .\ B[x]$ is the maximum universe among the universes of A and B w.r.t. the subtyping relation.*

Equality between types is understood in terms of propositional equality (Leibniz equality). It is introduced such that it complies to the Leibniz principle, i.e., two objects a and a' of the same type A are propositionally equal if and only if they are indistinguishable from each other with respect to all logical properties.

[10] Unlike Martin Löf type theory, sum types are not logical propositions and cannot represent the logical existential quantifier.

3.2. Subtyping

Subtyping (or cumulativity) formalizes a subset relation between type universes, function spaces and Cartesian products built on top of universes. The rule (Sub) is the basic subtyping relation over terms. Notice that according to this rule, a given term may have several types. However, it can be shown that whenever a term is typeable, it has a uniquely determined principal type (see, e.g., Luo (1994) for more details). This principal type is the minimum type of the term with respect to the (Sub) rule and relatively to a given context. As a particular case of the (Sub) rule, a (Conv) rule says that two terms are convertible if their corresponding types are equal.

$$\frac{\Gamma \vdash M \ : \ A \quad \Gamma \vdash A' \ : \ Type \quad A \preccurlyeq A'}{\Gamma \vdash M \ : \ A'} \ (\mathsf{Sub}) \qquad \frac{\Gamma \vdash M \ : \ A \quad \Gamma \vdash A' \ : \ Type \quad A \simeq A'}{\Gamma \vdash M \ : \ A'} \ (\mathsf{Conv})$$

The (Conv) rule has an intensional nature and reflects what is called the *computational* equality. However, this result does not rule out its (implicit) extensional meaning, i.e., the fact that if two types are computationally equal, then they have the same objects.

Lemma 3. *Luo (1994) The subtyping relation is a partial order over terms with respect to conversion.*

The following meta-theoretic property from Luo (1994), introduced as a corollary, will be useful for our purpose:

Corollary 1. *Luo (1994) The relation $\preccurlyeq$ is the smallest partial order over terms with respect to conversion such that:*

1. *if $A \simeq A'$ and $B \preccurlyeq B'$, then $\Pi x : A \,.\, B \preccurlyeq \Pi x : A' \,.\, B'$*
2. *if $A \preccurlyeq A'$ and $B \preccurlyeq B'$, then $\Sigma x : A \,.\, B \preccurlyeq \Sigma x : A' \,.\, B'$*

The logical consistency[11] of the core theory has been demonstrated and the decidability property has been deduced as a corollary Luo (1994); Werner (2008). In the K-DTT type theoretical layer, the usual logical operators follow the syntax of higher order logic Prawitz (1965); Coquand (1985) and can be expressed using the (dependent) product. For example, the logical implication $A \supset B$ is expressed with the non-dependent product $A \rightarrow B$[12] while the logical connectives, i.e., the conjunction $A \,\&\, B$, the disjunction $A \vee B$ and the negation $\neg A$ are respectively defined with $\Pi R : Prop \,.\, (A \rightarrow B \rightarrow R) \rightarrow R$, $\Pi R : Prop \,.\, (A \rightarrow R) \rightarrow (B \rightarrow R) \rightarrow R$ and $A \rightarrow \bot$.

4. K-DTT: the Ontological Layer

To build the ontological layer of K-DTT, we have considered the DOLCE ontology of particulars Masolo-et-al (2003); Gangemi-et-al (2002) because (i) it adopts both a descriptive and multiplicative[13] approach which does not classify universals leaving room for conceptual choices about universal structures, (ii) it is designed to be minimal, in that it includes only the most reusable and widely applicable upper-level categories and (iii) it remains neutral about the spatio-temporal properties (see Masolo-et-al (2003) for more details) and thus, provides a significant level of interoperability. The logical part of DOLCE is replaced here with a core type theory with the consequence that all relations defined in DOLCE are expressed in K-DTT using this core (for example, the hierarchical relation is_a is derived from the subtyping rule). The hierarchical taxonomy of particular categories will serve as a backbone, referred to as DOLCE backbone, for defining the stratification which allows more fine grain categories in the final domain ontology. Finally, the DOLCE backbone plus ontological commitments on appropriate structures

[11] For a type theory, the logical consistency is identified with termination.

[12] The "equivalence" between the logical reading $\supset$ and the computational reading $\rightarrow$ is a consequence of the Curry-Howard isomorphism.

[13] Different entities can be co-located in the same spatio-temporal location.

expressed within type theory will form the ontological layer of the K-DTT theory. It can be used to express knowledge as long as the added features respect the core structures together with their logical constraints (see e.g., Dapoigny (2010a)).

4.1. Expressing Knowledge with K-DTT

Most formal ontologies refer to universals and particulars which are further refined in subcategories (e.g., relations). Universals behave like general invariant patterns while types correspond to the result of a categorization procedure, hence types have a natural adequacy with universals. However, it is worth noticing that the term "type" denotes here type in the sense of type theory, that is, it covers a broader spectrum that its usual meaning in philosophy as well as in object-oriented languages. K-DTT objects are equipped with meanings using the Curry-Howard isomorphism in such a way that any type expressing some knowledge corresponds to a logical formula, therefore it behaves as a theory of reasoning and knowledge. The constructive semantics asserts that a type gains meaning (i.e., justification) from its constructor. In other words, a type in the K-DTT lower layer becomes a category in the upper layer. Then, deciding a category for a proof object corresponds to assigning ontological meaning to the object. We assume that categories are intensional entities and that ontological classes are expressed by categories representing universals as well as sets of particulars. Notice that we can go further the usual notion of type with dependent types in which we can mix instance-level and type-level information.

In the K-DTT theory the abstract entities become represented by certain symbol configurations called terms. The concept of universe is replaced with the concept of ontological category and the cumulativity is related to the so-called subsumption (see subsection 4.7). What is termed category here refers to what is called concept in Description Logics, the main difference being that a category is expressible by a (dependent) type with all the capabilities that such a conceptual choice offers.

Definition 4. *Terms of the K-DTT upper layer are built up from a set $\mathcal{V}$ of variables together with a set of categories $\mathcal{C}at$ including a subset of primitive categories $D_B = \{Prop, Thing, PT, AB, R, TR, T, PR, \ldots, STV, ST, PRO\}$ (where the sequence $PT, ..., PRO$ refers to the DOLCE set of basic classes), and the following category forming sets:*

- $\mathcal{P}_R$ $::= \Pi\mathcal{V} : \mathcal{T}\ .\ P$ *with* $\mathcal{T} \in \mathcal{C}at \cup \mathcal{R}el$ *for rigid property types,*
- $\mathcal{P}_{AR}$ $::= \Sigma\mathcal{V} : \mathcal{T}\ .\ \mathcal{C}_{AR}$ *with* $\mathcal{T} \in \mathcal{C}at \cup \mathcal{R}el$ *for anti-rigid property types,*
- $\mathcal{R}el$ $::= \Sigma\mathcal{V}_1 : \mathcal{T}_1\ .\ \ \ldots\ .\ \Sigma\mathcal{V}_n : \mathcal{T}_n\ .\ P$ *for n-ary relation types,*
- $\mathcal{C}$ $::= \Sigma\mathcal{V}_1 : \mathcal{T}_1\ .\ \ \ldots\ .\ \Sigma\mathcal{V}_n : \mathcal{T}_n\ .\ P$ *for context types and,*
- $\mathcal{R}ol$ $::= \Sigma\mathcal{V} : \mathcal{T}\ .\ \mathcal{C}$ *with* $\mathcal{T} \in D_B \cup \mathcal{R}ol$ *for role types,*
- $Sortal$ $::= \Sigma\mathcal{V}\ :\ \mathcal{T}\ .\ IC$ *with* $\mathcal{T} \in D_B \cup \mathcal{R}ol$ *for sortal types.*

where $\mathcal{T}$, $\mathcal{T}_i$ stand for terms, $\mathcal{C}_{AR}$ for an anti-rigidity criterion, IC for an identity criterion and P denotes some predicate depending on appropriate arguments. Any domain ontology can be derived from this ontology by adding sub-categories of some categories $K \in \mathcal{C}at$.

Definition 5. *A term $\mathcal{T}$ is well-formed if for some foundational ontology $\mathcal{O}$ providing the environment, we have $\mathcal{O} \vdash \mathcal{T} : K$ for some category $K \in \mathcal{C}at$.*

For example, the judgment $LegalPerson : SAG \vdash x : LegalPerson$ asserts that the variable x belongs to the category $LegalPerson$ provided that it is a well-formed term, i.e., $LegalPerson : SAG$ (Social AGent). In the following, we often forget the environment for the sake of clarity, however it must be kept in mind that any assertion is relative to an explicit environment constituted by the foundational ontology and involving all the arguments of the new defined type.

All ontological structures are defined and checked within the Coq proof-assistant Coq Development Team (2010). Already used for proving e.g., reliable knowledge-based system design Calegari-et-al (2010), relational database management systems Malecha-et-al (2010) and software certification Bruni-et-al (2007), the Coq proof assistant will serve as a platform for proving the ontological assertions. The Coq

system results of over ten years of research at the INRIA and provides a proof assistant whose underlying metalanguage is the Calculus of Inductive Constructions that has been extended with an infinite hierarchy of universes (CC^{ω}). The core of the system is a proof-checker allowing the declaration of axioms and parameters, the definition of any types and objects and the explicit construction of proof-objects represented as lambda-terms. The interactive theorem prover executes tactics written in Caml[14] and proofs can be built progressively by means of this tool in a top-down style that generates subgoals and backtracking as needed.

4.2. Representing Ontological Classes

The two basic entities of ontologies, i.e., particulars and universals are respectively understood in terms of proof objects and categories in K-DTT. Universals are described (i) with primitive categories, (ii) with property types and (iii) with relation types. Primitive categories are in line with the existence of "natural types", i.e., they can be identified as types in isolation (see e.g., Sowa (1988)). What is called a property type here, is close to the relational moment type Guizzardi (2005) which is existentially dependent on other particulars. Let us consider how a moment universal can be accounted for in K-DTT. We only consider here (for the sake of simplicity) an intrinsic moment which uniquely depends on a single particular. The inherence relation is isomorphic with type dependence on values. An electric charge is an example of a moment universal since this charge only exists if there is a conductor which supports it. It follows that a moment universal can be described with a sum type where the dependent family, i.e., the electric charge depends on a value having the category $Conductor$. The example is summarized in figure 1 in the context of the so-called Aristotelian Square. Property types are divided into mandatory properties (rigid properties)

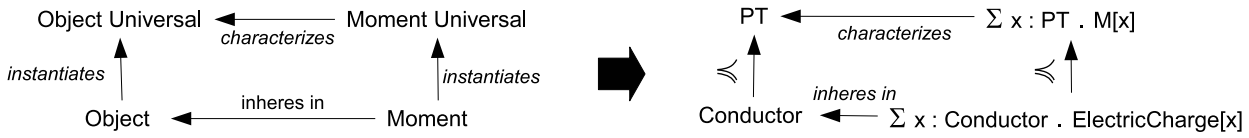

Fig. 1. Revisiting the ontological square in K-DTT.

and possible properties (anti-rigid properties). Two important subcategories of anti-rigid properties are role types and sortal types. A role type includes a context type itself arranged with relation types. Role types will be discussed in detail in section 6. All K-DTT structures and their corresponding classes in a foundational ontology are reported on figure 2. The DOLCE backbone classifies categories of particulars such as APO (which stands for Agentive Physical Object) and describes the category of these particulars. The categories of particulars which belong to the DOLCE backbone are described with a K-DTT primitive category where each position of the category in the backbone is computed automatically by the Coq theorem prover. We assume that a category is never subordinate to more than one higher category within the backbone. Even if in some cases there are counterexamples of this, it is however always possible to remove these overlapping categories Guarino (2002). Each time we are interested in a certain collection of categories that share some common properties, then universes can be introduced. Every category that belongs to a universe is considered as an object of upper universes. For example, in the Coq proof assistant, the compiler generates for each new category which is an instance of "Type" a new index for the universe and checks that the constraints between these indexes can be solved. In summary, primitive categories of particulars are the basic components from which all typed structures are built. The following fragment of Coq code describes an excerpt of the DOLCE backbone.

[14] see the site http://caml.inria.fr/index.en.html.

```
Definition Thing                := Type.
Definition PT           : Thing := Type.
Definition ED           : PT    := Type.
Definition PD           : PT    := Type.
Definition Q            : PT    := Type.
Definition AB           : PT    := Type.
                        ...
Definition SAG          : ASO   := Type.
Definition SC           : ASO   := Type.
Definition Person       : APO   := Type.
```

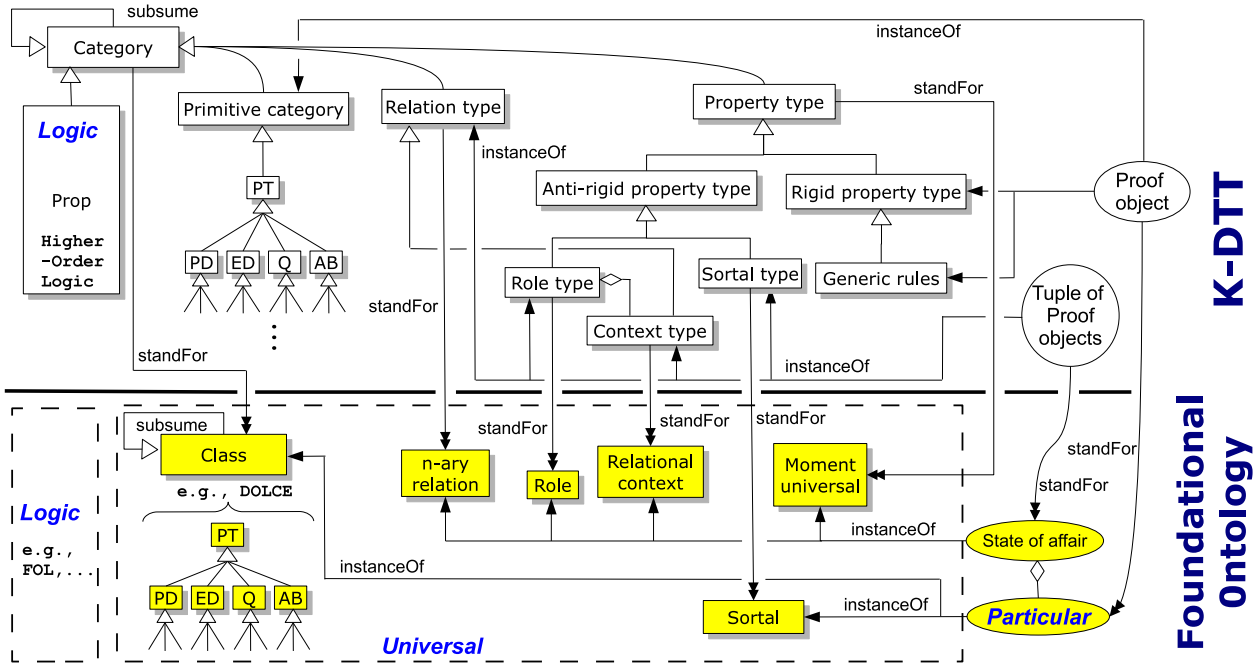

Fig. 2. Ontological categories in K-DTT.

These definitions will be implicitly included in further code fragments. The highest level is assumed for the category $Thing$ which covers all categories while PT stands for the type of primitive categories for particulars. Then all other types are ordered according to their place in the DOLCE backbone. Since it has only a type affectation, the ontological class "person" is described by the primitive category $Person$ in $Person : APO \vdash x : Person$ and may correspond e.g., to the proof (instance): $x = JohnDoe$ while $Person : APO$ asserts that $Person$ is well-typed. The relation of instantiation (:) between a universal and its instance corresponds here to type inhabitation. Unlike FOL-based ontologies, it describes in a natural and simple way the relation between universals and particulars without the need to introduce specialized ad hoc formulations. For example, in Smith (2004), the author suggest a formal theory of Is_a and $Part\_of$ based on a relation of instantiation between an instance and a class. This relation needs axioms governing its use while in K-DTT, the relation of instantiation is already part of the theory and does not require any further axioms.

The equality between terms in the ontological layer is ascribed to be coherent w.r.t. the Leibniz equality of the lower layer. It relates to the usual definition of the identity condition for an arbitrary property P, i.e., $P(x) \wedge P(y) \rightarrow (R(x, y) \leftrightarrow x = y)$ with a relation R satisfying this formula. This definition is carried out for any type in K-DTT since equality between types requires the Leibniz equality (see subsection 4.1). The major reason is that identity can be uniquely characterized if the language is an higher-order language in which quantification over all properties is possible Noonan (2011). This higher-order property yields

that Leibniz's Law, which is at the basis of identity in the lower layer of K-DTT is expressible in this language.

4.3. Representing Propositions, Facts and Predicates

At the ontological level, a proposition is a sentence expressing that something has/can have a truth value, expressing whatever it expresses. Let us consider the sentence "this ball is red". If this sentence is true, then it is in virtue of the being red (the redness) of this ball, and if no such redness exists, then "this ball is red" is false. The redness we are speaking of here is a particular which depends on the ball, and not a universal redness shared by all red things. Dependent types easily address this kind of subtlety with the dependent family $Red[a]$ with a a particular such that $a : Ball$: $Red[a]$ is a (dependent) category which denotes here the redness indexed by the particular ball a. In set theoretic frameworks, a proposition is the set of possible worlds in which it is true. Type theory treats propositions as primitives rather than treating them as sets of worlds (the so-called possible-worlds semantics) since the notion of truth in a possible world is replaced by the notion of proof. Therefore, each proposition is represented with a type, the type of its proofs.

Definition 6. *Every type P of the category $Prop$ is called a proposition. If a term p_1 is of the type P, then p_1 is called a proof*[15] *for P.*

As a consequence, propositions are seen as categories (types of proofs) and the semantic value of a predicate is roughly the set of individuals (i.e., proofs) that bear it. For instance, we can find a proof (e.g., resulting from a sensor measurement) for the variable p that proves it is raining with the judgment $raining : Prop \vdash p : raining$. Furthermore, we assume in the following part of the paper that the category $Prop$ is interpreted as the category of proof-irrelevant types, i.e., we cannot distinguish the different proofs of a given proposition.

Facts are akin to states of affairs in the world. According to Russell (1986) p. 163, a fact is "[...] the kind of thing that makes a proposition true or false". A constructive reading will change this assumption in "a fact is the kind of thing that proves a proposition". Therefore, in the ontological layer of K-DTT they are concrete entities of the world which we can speak about and they refer to an instantiation of universals. In the lower layer, they are proof objects which instantiate types.

The other kind of primitive in language are n-place predicates (with $n = 1, 2, ...$), which correspond to relations. A predicate is the result of combining n names with an n-place predicate. They can represent n-ary relations at the ontological level while at the logical level, predicates are relations whose output is the $Prop$ category.

4.4. Representing Relations

The strength of dependent types in type theory allows to express relations as primitives of the language. The first consequence is that they are terms of the logic and can be involved in complex predicates. At the ontological level, relations are hierarchical (e.g., subsumption or part-of relations[16]) or non-hierarchical (e.g., domain relations). They denote tuples involving particulars (the last type of the rightest term of the tuple is generally $Prop$). Tuples correspond to sum type structures having more than a single argument. While binary relations require two arguments, relation types can generally offer as many arguments as needed. For example, the relation type:

$$\Sigma x : Person\,.\,\Sigma y : Car\,.\,Own[x, y]$$

where $Person$ and Car denote sortals and $Own : Person \rightarrow Car \rightarrow Prop$ stands for the relation in which a persons owns a car. Proof objects for such a relation type could be[17]

[15] The term proof object is restricted to proofs for categories.

[16] We refer her to the part-of relation which is transitive by contrast with the partonomic relation which is usually not (see Dapoigny (2010a) for more explanations).

[17] Proof objects for the variable of type Car are license registration numbers as extracted from a database.

$\langle JohnDoe, \langle AZ787EK, q_1\rangle\rangle$, $\langle MikeThumb, \langle BC125AB, q_2\rangle\rangle$, etc. with q_1, q_2, ... the respective proofs for $Own[JohnDoe, AZ787EK]$, $Own[MikeThumb, BC125AB]$, etc. In this example, with $s \triangleq \langle JohnDoe, \langle AZ787EK, q_1\rangle\rangle$, we can easily access the components of the proofs with $\pi_1 s = JohnDoe$ and $\pi_1\pi_2 s = AZ787EK$[18].

A generic n-ary relation type can be constructed according to the following definition.

Definition 7. *(N-ary relation) A n-ary relation type Rel between n categories is expressed with a sum type having these categories as arguments and whose extension consists of all the proofs for that relation.*

$$Rel \triangleq \Sigma\, x_i : T_i\ .\ \ldots\ \Sigma x_{i+n} : T_{i+n}\ .\ R[x_i, \ldots, x_{i+n}] \tag{1}$$

with $R\ :\ T_i \rightarrow \ldots \rightarrow T_{i+n} \rightarrow Prop$ *and* $Rel : max(T_i, \ldots, T_{i+n})$.

in which T_i denote terms of the K-DTT language as described in subsection 4. This definition is general and applies at different levels of abstraction w.r.t. subtyping. For example, let us mention the binary relation "ParticipateIn" described in Keet (2008). In the relation type, the term $ParticipateIn$ (which belongs to the $Prop$ category) relates a first argument that must be an endurant (ED in DOLCE) together with a perdurant (PD). As explained in subsection 2.5, these arguments belong to different category levels, then according to basic rules of subtyping (Luo (1994) and Coq Development Team (2010)), the type of the whole relation must be at least in the highest level among components of the relation type.

$$(\Sigma x : ED\ .\ \Sigma y : PD\ .\ ParticipatesIn[x, y])\ :\ PT$$

since ED and PD are subcategories of PT.

Some particular relations such as the so-called is_a and $part\_Of$ relations have established themselves as foundational to domain ontologies as well as to foundational ontologies (e.g., DOLCE). While it is commonly assumed in the literature of knowledge representation that the relation is_a can be seen as the set inclusion relation, we depart from this assumption and rather consider an intensional approach (this aspect will be developed for role types in section 7). The $part\_Of$ relation can be generalized through the notion of parameterized type (see Dapoigny (2010a) for more details).

4.5. Representing Properties

Properties are entities that can be attributed to things or predicated of them and then it can be said that objects exemplify properties. In K-DTT, properties correspond to moment universals since their existence depends on some external entity. We first introduce rigid properties which are expressed with product types whose argument is a relation type taking itself categories as arguments. It yields a type which belongs to $Prop$. For instance, to represent the fact that any apple is colored, one may introduce:

$$\Pi u : (\Sigma x : Apple\ .\ Color[x])\ .\ HasColor[\pi_1 u]$$

in which $Color$ is a dependent type (i.e., a function $Apple \rightarrow Color$) which for each instance x (e.g., $ThisApple$) of the type $Apple$ yields a type (e.g., $Color[ThisApple]$) whose proof objects range over the color domain, e.g., $yellow$. Conversely, $HasColor$ is a predicate having the type $Apple \rightarrow Prop$ whose proof is a truth value. In other words, it means that all apples have a color (universal quantification). Another interesting benefit of considering properties as (dependent) products is that variables can range over them allowing to quantify over properties.

More complex and contingent properties can be described by considering a sum type as argument. A generic complex structure may require an argument of type relation (possibly nested) R:

$$RigidProp\ \triangleq\ \Pi u : R\ .\ Pr[\vec{\gamma_1}u, \ldots, \vec{\gamma_n}u]$$

where $n \geq 1$ and Pr, of the type $T_1 \rightarrow \ldots T_n \rightarrow Prop$ with $\vec{\gamma_i}u : T_i$. The relation type R can be nested up to a finite number of levels and each $\vec{\gamma_i}$ stands for a sequence of π_1 or π_2 relative either to variable u

[18]Notice that π_2 is first applied to s, then π_1 is applied to this result.

for nested sum types or to an empty sequence for categories. One can e.g., express the fact that patients which suffer of Alzheimer disease lack autonomy:

$$\Pi u : (\Sigma x : Patient\ .\ \Sigma y : AlzheimerDisease\ .\ SufferFrom[x, y])\ .\ NotAutnomous[\pi_1 u]$$

Alternatively, anti-rigid properties are possible properties relating an entity (who has the property) with an anti-rigidity criterion. An anti-rigid property type $AntiRigidProp$ is expressed as a relation type having as left term a type, either a category or a relation, called the subject S of the property and as right term, a type C_{AR} which defines the criterion to be a type having this property:

$$AntiRigidProp \triangleq \Sigma x : S\ .\ C_{AR}[x] \quad \text{with: } C_{AR}[x] \text{ an anti-rigidity criterion type}$$

Let us consider the example of a student within a university. One of the basic characteristics of the anti-rigid criterion is the fact that any student must be registered with the corresponding definition:

$$Student \triangleq \Sigma x : LegalPerson\ .\ \Sigma u : Univ\ .\ RegisteredIn[x, u]$$

and $RegisteredIn\ : LegalPerson \rightarrow Univ \rightarrow Prop$ With the DOLCE categories SC for Society and SAG for Social Agent, the environment Γ assumes that $Univ : SC$ and that $LegalPerson : SAG$ since we are interested here in a social anti-rigid property. Notice that rigid properties are automatically applied to non-dependent types (using the Leibniz law for types) while anti-rigid properties require sum types to formalize the subset relation over types.

4.6. Representing Meta-properties

In K-DTT any term (i.e., anything) can possess properties. As a consequence, a property may have properties. A meta-property which is the property of a class in foundational ontologies, is described in a uniform way i.e., the same syntax as for properties of particulars. For example, as shown in subsection 4.5, distinct meta properties can be described with appropriate types. For example, rigid properties are described with product types since they require a $\forall$ quantifier while anti-rigid properties rather involve sum types which approximate subset types (i.e., the proof objects for the category which is concerned by the anti-rigid property is a subset of the proof objects for the category). Let us consider the sortal property based on the presence of one or more IC (Identity Criterion) Guarino (2000). To state that a particular $x : PT$ is an instance of a sortal type, we assume that not every subtype of the category PT may be of the (meta)type sortal and to fulfill this constraint, we have to consider the category $Sortal$ as an anti-rigid property:

$$Sortal \triangleq \Sigma x : PT\ .\ IC[x]$$

where IC is a dependent property which formalizes an identity criterion. Any sortal s should be subsumed (i.e., a subtype-of) by this definition while its instances are pairs $\langle pt, id\rangle$ where id is a proof object for $IC[pt]$. It follows that in all its (proved) instances, pt necessarily satisfies the criterion $IC[pt]$. It is the type-theoretical translation of the assumption "a property P carries an identity condition C if all its instances necessarily satisfy C". We can assert for example that a person is identified with its National Registration Identity Card (NRIC). This can be captured by the property $Citizen \triangleq \Sigma x : Person\,.(\Sigma n : NRIC\,.SignatureOf[n, x])$ in K-DTT. A proof for this sortal could be $\langle JohnDoe, \langle 32270005187T, p1\rangle\rangle$ with $p1$ a proof for $SignatureOf[32270005187T, JohnDoe]$. If we know that the considered person is JohnDoe and that the NRIC is what can be read on its card, we prove respectively $Person$ and $NRIC$ but if we obtain the information that the card is a fake, then there is no proof for $SignatureOf[32270005187T, JohnDoe]$ and the identity criterion fails. Then, only a subset of all particulars will also have the type sortal (those for which an identity criterion exists) and the identity criterion will propagate in the subsumption only if every term of the relation type is in the subtyping relation. Let us illustrate these aspects with an example extracted from Guarino (2002). If we consider the category $TimeDuration$ whose proof objects are e.g., $two\ hours$ and a category $TimeInterval$ defining particular intervals of time, such as $4:00-6:00,\ 2012/17/2$. One can think of $TimeInterval$ as a subclass of $TimeDuration$, since all time intervals can be considered as time durations. What about

the identity criteria for such sortals? Assuming that two durations are identical iff they have the same length and that two intervals are the same iff they have the same length and they occur at the same time $t : StartTime$, the respective properties for the identity criterion have the form:

$$TimeDuration \triangleq \Sigma t : TemporalRegion\ .\ Length[t]$$
$$TimeInterval \triangleq \Sigma t : TemporalRegion\ .\ (\Sigma x : StartTime\ .\ LengthAt[t, x])$$

These two sortals describing particular time durations and intervals are not related by subtyping w.r.t. corollary 1 while their left terms are. It follows that the subsumption does not hold since their identity criteria are obviously incompatible.

4.7. Expressing Subsumption

Subsumption consists in deciding whether one category is "more general than" another one and the subsumption relation A subsumes B says that being a B logically implies being an A. The notion of subsumption has several readings whose the more important ones are extensional and intensional Woods (1991); Napoli (1992). There are some drawbacks to the extensional interpretation of subsumption because (i) determining whether the extension of one concept is included in the extension of another one is often undecidable and (ii) observing that two concepts have the same extension does not mean that they are identical. Furthermore, in the intensional reading, a concept subsumes another concept only if this result can be inferred from the examination of the internal structure of this concept involving more domain dependent inferences.

In the (intensional) theory K-DTT, we assume that the identity of two categories holds iff their proof objects cannot be distinguished by any property expressed by a predicate over the category of these objects. This ontological assumption complies with many approaches in formalizing ontologies and is a direct translation of the Leibniz law (see subsection 3).

Definition 8. *Two categories are identical iff they have the same properties.*

Unlike relation types described with sum types, in K-DTT, the subsumption relies on the subtyping which is explicitly described in the type-theoretical layer by the (Sub) rule (see 3.2). Notice that this definition of the subsumption yields a limitation restricting the possibilities of K-DTT to single inheritance. Subsumption-based hierarchies take advantage of the simple mechanism of the cumulative hierarchy of type universes. Every type that belongs to a category is considered as an object of upper categories. For example, for the categories $Sensor$ and $Device$, we can assert that $Sensor \preccurlyeq Device$ and that $Device$ subsumes $Sensor$ since a sensor has, at least, all the properties of a device and then, contains more precise information than a device. In that precise case, properties are implicit, but they can be explicit having the form of product types using lemma 4. This characteristics refers to the subsumption or is_a hierarchy in ontologies and relates to the subtyping relation as follows.

Lemma 4. *Given two categories T and T', such that T subsumes T', i.e., $T' \preccurlyeq T$, then any property that holds for T also holds for T':*

$$\lambda P : T \to Prop\ .\ \forall x : T\ .\ \forall y : T'\ .\ (Px) = (Py)$$

with the corresponding Coq code:

```
Definition Cat1                       := Type.
Definition Cat2         : Cat1        := Type.
Hypothesis Property     : forall x : Cat1, Prop.
Lemma subsume           : (forall (x : Cat1) (y : Cat2), Prop).
Proof.
exact ((fun (x:Cat1) (y:Cat2) => Property x = Property y)).
Qed.
```

The is_a hierarchy of DOLCE is represented by a cumulative hierarchy of categories where each node in the tree is assigned a universe (see subsection 4.2), the highest universe being $Thing$ the category of all categories. It follows that the DOLCE backbone is included in a core hierarchy of universes whose upper universe is always $Thing$ while the lowest universe depends on the user's choices. For instance, let us consider some basic categories such as $Eating$, $Reading$, $ChemicalReaction$, etc. that we expect to relate to the DOLCE backbone of type universes. The categories $Eating$, $Reading$, $ChemicalReaction$ are specialized types of perdurants. Then, the smallest level is assumed to be a universe including (at least) $Eating$, $Reading$ and $ChemicalReaction$ whereas this universe is referred to as PRO (processes). At the same time PRO is a subtype of stative perdurants (STV) themselves subtypes of perdurant types PD. It follows that:

$$Eating \preccurlyeq PRO \preccurlyeq STV \preccurlyeq PD$$

The same relations hold for $Reading$ and $ChemicalReaction$. In other words, $Eating$ represents all processes consisting in eating something, PRO denotes all processes, STV focuses on all stative perdurants while PD addresses all categories of perdurants. However, each new type insertion within this classification preserves the original order but will automatically increment each universe index for the upper levels. For example, if one supposes that $Eating$ belongs to the universe $Type_i$ and that there exist different kinds of $Eating$, like $StuffOneself$, $Devour$, $NibbleAt$, etc., then the insertion of these categories (which are subsumed by $Eating$) will assign them in the universe $Type_i$, $Eating$ to $Type_{i+1}$, etc. One will write:

$$\begin{array}{lcl} StuffOneself & \preccurlyeq & Eating \\ Devour & \preccurlyeq & Eating \\ NibbleAt & \preccurlyeq & Eating \end{array}$$

It results that the previous hierarchy of universes is shifted one level higher with the introduction of this new level and preserves the previous classification of DOLCE categories. Notice that subsumption in FOL can be directly expressed by a logical implication, which is illustrated in the translation of the following fragment. The logical formula $\forall x : (Devour[x] \rightarrow Eating[x])$ where the variable x ranges over all domain objects will express that everyone which is devouring is also eating. In K-DTT, subsumption operates at the level of types (and therefore, of categories), that is types are considered as objects which can be more easily manipulated and reasoned about. For example if a type represents a complex relation instead of a simple category like $Eating$, then generic rules can be applied and enhance the accuracy of the reasoner accordingly.

Subtyping ($\preccurlyeq$) must not be confused with typing (:). In the subtyping $B \preccurlyeq A$, the category A details the necessary conditions to be a B whereas category definition (i.e., $A \triangleq ...$) refers to the necessary and sufficient conditions to be an A. For example $LegalPerson : SAG$ considers that a (legal) person has *all* the properties of the category social agent (SAG). The DOLCE backbone asserts that categories may be in the subtyping relation like in $SAG \preccurlyeq ASO$ (ASO stands for the "agentive social object" category) meaning that SAG has *at least* all the properties of ASO. These considerations are extended to relation types. Let us consider the description of a necessary and sufficient condition in K-DTT. It would require a relation depending on a single argument.

$Hepatitis \triangleq \Sigma x : InflammatoryDisease\,.\,Liver[x]$ and: $Hepatitis : HasLocation$

Let us now consider a necessary condition. A relation type can be defined as the subtype ($\preccurlyeq$) of a generic one. It follows that the above expression could be written:

$HL1 \triangleq \Sigma x : Inflammation.Endocardium[x]$ with: $HL1 : HasLocation$
and $EndoCarditis \preccurlyeq HL1$

Notice that the subtyping is only implicit in Coq and, in order to prove that this relation holds between A and A', it is first assumed that the hypothesis $M : A$ holds and then it is checked whether $M : A'$ is proved. If it is the case, then the subtyping $A \preccurlyeq A'$ holds. The Coq syntax reflects the above differences in the following way. The definition statement e.g., `Definition PD : PT := Type.` means that we are defining a new universe PD which is a subtype of the existing universe PT, that is, $PD \preccurlyeq PT$. Alternatively, if

the Coq syntax states that `Hypothesis x : LegalPerson.`, it means that the new variable x is of type $LegalPerson$, i.e., $x : LegalPerson$ provided that $LegalPerson$ has already been defined (well-typed) e.g., $LegalPerson \preccurlyeq SAG$.

4.8. Generic Rules

A generic rule uses a product type to infer a new relation from one or more existing ones. Let us consider for example the right distributivity introduced in Artale-et-al (1996). The right distributivity means that a relation on a part may distribute its related predicate to the whole. Let us explain this rule by considering first a generic parametric[19] relation type $hasLocation$, saying that a process occurs in some Physical EnDurant (PED). Second, we should introduce a generic part-whole relation $PartOf$ expressing that some physical endurant is part of some other physical endurant (the whole). However, to express that the part in the first relation type refers to the same object than the one which occurs in the part-whole relation type, we rather introduce an appropriate generic relation type[20] $HLPO$ which replace the dependence on an arbitrary term of type PED with the output of the $HasLocation$ relation type, i.e., a term whose resulting type is also PED. This is required to be sure that we act on the same proof object.

$$HasLocation[x : PRO,\ y : PartPRO[x]] \quad \triangleq \quad \Sigma x : PRO\ .\ PartPRO[x]$$
$$HLPO[u : HasLocation[],\ z : Whole[\pi_2 u]] \quad \triangleq \quad \Sigma x : HasLocation[]\ .\ Whole[\pi_2 x]$$

with $PartPRO : PRO \rightarrow PED$ and $Whole : PED \rightarrow PED$. For more clarity, we do not provide the explicit arguments of parametric types since they are given in the code below. If a part is right-distributive, then the relation which holds for the whole is also true for the parts, i.e., more formally, the generic rule admits a proof object for the relation type $HLPO$ as input, and outputs a proof object for the updated relation type $HasLocation$:

$$\Pi z : HLPO[]\ .\ HasLocation[][\pi_1\pi_1 z,\ \pi_2 z]$$

The Coq code looks like:

```
Definition PartPRO(p:PRO)                 : PED             := Type.
Definition Whole(pa:PED)                  : PED             := Type.
Definition HasLocation(x:PRO)(y:PartPRO x)                  := {x:PRO & (PartPRO x)}.
Definition HLPO(u:{x:PRO & (PartPRO x)})(z:Whole (projT2 u)) := {u:{x:PRO & (PartPRO x)}
                                                                & Whole (projT2 u)}.
Axiom RightDistrib : forall (z:{u:{x:PRO & (PartPRO x)} & Whole (projT2 u)})(v:PartPRO
                (projT1 (projT1 z)))(w:Whole v), HasLocation (projT1 (projT1 z)) w.
```

Notice that with the intermediary variable v, the variable w stands for $\pi_2 z$. Substituting variable $\pi_1\pi_1 z$ in $HLPO$ with $Activity$, $PartPRO[\pi_1\pi_1 z]$ with $Room$ and $\pi_2 z$, i.e., $Whole[PartPRO[\pi_1\pi_1 z]]$ with $Building$, it follows that the corresponding proof object for $HLPO$ expresses that an activity is located in a room itself located in a building. Applying the generic rule yields that $HasLocation()[Activity, Building]$, proving that the activity is also located in the building. When similar generic rule types are defined they can be applied for all relations types as far as they are in the subsumption relation with each generic rule type.

5. Modeling Roles with Dependent Types

5.1. Related Works

While the first part of the paper has presented a more expressive language for expressing ontological knowledge, this part will focus on the formalization of roles. They are also of interest in closely related

[19] Unlike usual dependence, the dependence in the parametric type is on external variables to the environment.
[20] These types are parametric for more flexibility.

domains such as knowledge engineering, object-oriented and conceptual modeling, multi-agent systems, philosophy, cognitive semantics and linguistics since they may introduce relevant issues or solutions for a role model. More precisely, the concept of role has received much attention in the conceptual modeling and object-oriented modeling literature Steimann (2000); Wieringa-et-al (1995), in knowledge representation Guarino (1992); Sowa (2000); Welty (2001); Masolo-et-al (2005); Guizzardi (2005), in knowledge engineering van Heijst-et-al (1997) and (semi-)formal languages such as the Unified Modified Language (UML) Rumbaugh-et-al (1999) or Description Logics (DL) Baader-et-al (2003). There has been extensive theoretical research about roles but no consensus on its understanding appears so far. Even a widely used ontology language like OWL does not consider roles as a primitive. Furthermore, it is clear that an improper modeling of roles would have a great impact on truth maintenance along *is-a* hierarchies Guarino (1998). Despite these difficulties, some basic considerations appear to be admitted, such as the special relationship of roles to other objects (or roles) which are said to play the role Masolo-et-al (2005); Loebe (2007); Mizoguchi (2007). In such a fundamental scheme, players are entities playing a role within a given context. This general characterization could be captured by the first-order axiom Loebe (2007):

$$\forall x\,(Role(x)\ \leftrightarrow\ \exists yz\,(plays(y,x)\ \wedge\ roleOf(x,z))) \tag{2}$$

where $roleOf$ denotes the relation between roles and contexts. Let us recall the well-known student example in which the role $Student$ is associated with players of type $Person$ in the role specification under the context, e.g., of a university.

Recently, Boella-et-al (2007) has isolated a small number of consensual role characteristics which cover the fields of knowledge representation, conceptual modeling and object-oriented modeling. First, roles have a relational and interactional nature. This aspect is pointed out in Noble (2010) which claims that a role makes sense only in the context of a relationship when the object is playing that role. Second, roles are associated to a social context within an organizational-centric perspective and third behavior appears as a main feature of roles, e.g., through the temporal aspect. Such a characterization is not sufficient and in order to provide a better understanding of the $plays$ and $roleOf$ relations, there are two problems to address (i) a review of the common theoretical issues to be solved and (ii) the expected characteristics that the concept of role must offer from a conceptual modeling (and ontological) perspective.

In Sowa (1988), the author first distinguishes between "natural concepts", i.e., concepts that can stand on their own, and "role concepts", i.e., dependent concepts, and second, asserts that role concepts are subsumed by natural concepts. Since a role concept is more specific than the natural concept and corresponds to a smaller extension, then this view seems appealing but yields subtle problems due to the dynamic nature of roles (see, e.g., Steimann (2000) for more details). This work has been developed in Guarino (1992) which adds the meta property of rigidity and the notion of founded concept. A concept is founded if all its individuals are related to another individual, excluding the part-of relation. For example, the concept $Student$ is necessarily related to the concept $course$. A concept is rigid if it contributes to the very identity of its instances. Conversely, a concept is anti-rigid if its properties are non-essential for all its instances. Above these assumptions, a concept is a role if it is founded and anti-rigid. According to this definition, $Student$ is a role since it is both founded (see above) and anti-rigid (e.g., if the individual $Johan$ is an instance of $Student$, $Johan$ can move in and out of the $Student$ role without loosing his identity). The introduction of founded types argues for a context-based framework for the role whereas the anti-rigidity has clearly a modal flavor.

The work of Masolo-et-al (2004) (in first-order logic) considers roles as properties that can themselves have properties while being linked to a context. The proposed formalism is based on reification and predefined primitives such as $DS(x)$ (x is a description), $DF(x,y)$ (concept x is defined by description y), etc. Embedding these primitives within the DOLCE ontology Masolo-et-al (2003), the basic categories of the ontology are in the ground level while social concepts and descriptions are in the (reified) conceptual level. While this approach appears as a significant contribution to understanding the ontological nature of roles, it suffers from some drawbacks. First, the use of reification is subject to discussion (viewed as a philosophically suspect mechanism Galton (1991); Akinkunmi (2000)) and second, the proposed formalism introduces a plethora of axioms making its assessment complex. This approach deserves credit

for improving the expressiveness of roles by introducing a meta-level of roles (e.g., predicates on social concepts). However, the price to be paid is an increasing complexity due to the first-order framework.

Alternatively, Loebe (2007) characterizes three kinds of role, i.e., relational roles, processual roles and social roles. Relational roles are roles that depend on other properties or roles while processual roles depend on processes supplied by the context. All these assumptions are translated in the syntax with first-order rules in which some constants refer to universals whereas others refer to individuals. Social roles require to adapt the relation between roles and contexts from "X $plays$ the $role$ Y in $roleOf$ C" to "X counts as Y in context C". However, due to the use of first-order logic, some difficulties emerge with the player-role-context approach such as i) a definition of the context which requires further elaboration, ii) a clear separation between representations of universals (e.g., with types) and individuals (e.g., with proof objects) and iii) the inability of classical approaches to exploit the meta-level status of roles.

In object-oriented and conceptual modeling, the focus is rather to find constructs able to represent roles in the context of classes and inheritance. Most works either concentrate on representational issues of roles Steimann (2000) or on object-oriented role models Wieringa-et-al (1995); Dahchour-et-al (2004). With respect to this perspective, various modeling issues are of interest, such as multiple inheritance, dynamic classification, object migration, etc. Most common ways of representing roles include named places Chen (1976), specification/generalization Van Paesschen-et-al (2005) or aggregates relating an object with its roles Steimann (2000). Difficulties arise in the specification/generalization approach with the *subtype-super type* paradox (see Steimann (2000) for more details). More precisely, viewing roles as Object-Oriented (OO) classes and assuming two classes X and Y, the state of objects of X is more general than the state of objects of Y, while at the same time the behavior of X is more specific than Y's behavior. As a consequence, roles can be super types (statically) and subtypes (dynamically), which is in contrast with the OO notion of inheritance where states and behavior are both inherited by subtypes. However, more recent works have addressed most of the problems which were inherent to OO programming (see for example, Herrmann (2007); Baldoni-et-al (2006)).

5.2. Towards Common Requirements

From the analysis of the role-based literature, a small number of common discussed features can be extracted, as already suggested in Steimann (2000); Loebe (2007). They can provide the basis for a suitable role representation.

1. The nature of roles, i.e., as universals or as particulars. As underlined in Loebe (2007), many approaches accept both role individuals and role universals.
2. The relation between roles and behavior. For that purpose, it seems appropriate to distinguish between properties which are required for having the role (context) with the properties which result (for the player) when the role is acquired. The behavior (of the player) will precisely depend on these resulting properties.
3. Role and identity, i.e., whether the identity of roles is different from their players.
4. Roles are modeled in terms of non-essential properties, and more precisely, of anti-rigid[21] unary predicates Welty (2001). For example, if we consider the $Student$ role, we can think of an entity moving in and out of the $Student$ role while being the same individual.
5. Roles are relationally dependent Masolo-et-al (2005), i.e., they require the existence of another entity. Furthermore, it is claimed that a role is specified w.r.t the nature of this entity within a context Mizoguchi (2007). For example, the role of student depends on the university in which he (she) is registered. This university is supposed to be a part of the current context. Furthermore, an entity can play a role concept only within a context Masolo-et-al (2004); Loebe (2007); Mizoguchi (2007) that is composed of external concepts whose ontological status is heterogeneous. This constraint can be formally addressed if the concept of context receives itself a formal definition. For example, in

[21] All objects having such a property can lose it without losing their identity.

Masolo-et-al (2004) argue for a cognitive context (i.e., a system of constitutive rules) while Guarino and Sowa associate roles to patterns of relationships.

6. Different entities can play the same role at the same time. This assumption results from the position of Sowa (2000) viewing role as properties. This aspect complies with the OO perspective but requires second-order relationships between types Steimann (2000).
7. An entity can play multiple roles during a specific time interval.
8. An entity can acquire and abandon a role (dynamic aspect). As a consequence, players can lose their membership in relations or social roles. This property clearly relates to the meta-property of anti-rigidity which is an extension of dynamicity towards modality Loebe (2007).
9. An entity can play the same role several times simultaneously Steimann (2000); Masolo-et-al (2004). In the latter, the author distinguishes between two levels, the ground level dealing with the basic classes of a ground ontology and the conceptual level dealing with (reified) social concepts and their descriptions. A structure embedding roles is a possible way to solve this difficult issue. However, if role types are assumed Mizoguchi (2007), then it depends strongly on the ontological nature of the type of role definition.
10. A role can play another role Steimann (2000). This open issue depends on the nature of the role definition. For example, in an OO framework it requires roles to have instances. For social roles (e.g., a given human playing an employee role which plays itself a project leader role), Masolo-et-al (2004) explains this situation by considering two roles on the same abstraction level. One social role playing another role is the idea addressed in Searle (1995) where the author uses iterated applications of the formula "X counts as Y". In spite of some rare solutions to this problem (e.g., Mizoguchi (2007)), it can be considered so far as an unsolved issue.

As underlined in Welty (2001), most features of a role model can be formally defined with meta-properties whereas some of the above features (e.g., 7, 8) also require different abstraction levels. Other works relate role universals with types Steimann (2000); Guizzardi-et-al (2004) but do not exploit thoroughly the type concept. None of the previous approaches has defined a clear separation between universals and particulars resulting in numerous problems of identity. Finally, the requirement for role hierarchies advocate for a subsumption mechanism able to address all the subtleties inherent to the multiplicity of role constraints. As a consequence, the interest in roles has led to the search for a common ground on which the different views of role (see above) can be judged and reconciled Masolo-et-al (2005); Mizoguchi (2007). With regard to this assumption, we propose a general modeling of role with the help of dependent types, that is, with a dependent type theory.

6. Roles as Dependent Types

6.1. Definition

In the ER model, roles are reduced to labels of types and make it impossible to construct role type generalization/specialization hierarchies which would provide better organized conceptual models. Among the role-based literature, some works adhere to the position of "roles as types" Steimann (2000); Guizzardi (2005). However, this very concept of type corresponds to the definition in use in object-oriented languages with a limited expressiveness. For example, the role data model suggested by Bachman (1980) has introduced role types as super types of entity types, which has led to difficulties (e.g., when regarding Student as a super type of Person it accounts for the fact that all persons can appear in this role and contradicts the dynamic viewpoint, namely that at any point in time only some of all persons existing at that time are students Steimann (2000)). Because of the position defended by Sowa (2000) and Guarino (1992) (among others), assuming that different entities can play the same role, a mere assignment of a role to a category would seem sufficient. However, it is claimed by many researchers that roles have a relational nature. According to Sowa, roles depend on additional properties via patterns of relationships. Alternatively, Guarino & Welty (Welty (2001)) rather claim that we need a kind of dependence based on

Husserl's concept of foundation, i.e., based on a generic existential dependence on external properties. This aspect can be expressed by means of what is known as dependent types. The last argument deals with the introduction of contexts. On one side, some works such as Searle (1995); Loebe (2007) refer to explicit contexts while on the other side, others consider patterns of relationships Guarino (1992); Sowa (2000). Since an explicit context is able to encapsulate heterogeneous types in which any of them can (i) express a relation or a sortal denoting a type of particulars, e.g., $University$ and (ii) depend on previous terms within the context structure, therefore the explicit introduction of context subsumes all approaches. In addition, the concept of context has been formally expressed in a variant of dependent type theory Dapoigny (2010b) with nested sum types arranged in a partonomic hierarchy. Since each context type is indexed on a player, their aggregation (for a given player) into wider context types automatically introduces several possible role types for that player. In summary, a formal representation of roles should (i) support a categorization mechanism, (ii) exploit dependent types, (iii) involve an explicit context definition and (iv) explain the connection between all kinds of roles and relation. The role structure proposed in K-DTT complies with all these constraints. We consider a role as a specific anti-rigid property type in which the context type depends on categories and related constraints introduced in its environment. The environment Γ provides the terms that are required for the context and that are used to constrain the type of the player in the following role structure:

Definition 9. *Given a category $K \in \mathcal{C}at$ and a context type $C \in \mathcal{C}$, a role type Rol is an anti-rigid property type having as left term a type $P : K$ called the player of the role and as right term, a context type C in which the player acts with:*

$\Gamma,\ x : P \vdash C : P \rightarrow \ldots \rightarrow Prop$, *then:* $\boxed{Rol \triangleq \Sigma x : P\ .\ C[x]}$

where the type of $Role$ is the maximum universe among all types occurring in the role expression.

Syntactically, the role context type C can range from a simple context type to complex nested context types. Proof objects of the type P are the players of the role and C is the context type which constrains P. Concrete roles $r : Rol$ are pairs $\langle M, N\rangle$ where each M is a proof object for a player (of the role) and N the related proof that the context exists (i.e., a predicate depending on the player). Semantically, the context does not merely represent the physical environment of an entity, but rather the way the entity perceives its environment. It follows that several entities can have different views on the same physical environment. If we consider for example, a student within the context of a university we assume that any student must be registered (at least once). With the DOLCE categories SC for Society and SAG for Social Agent, if the environment Γ incorporates the typing assumptions $Univ : SC$ and $LegalPerson : SAG$, then terms are well-formed. The following role definition:

$$Student \triangleq \Sigma x : LegalPerson\ .\ C[x] \tag{3}$$

introduces the context type $C[x : LegalPerson] \triangleq \Sigma u : Univ\ .\ RegisteredIn[x, u]$ with the predicate $RegisteredIn\ : LegalPerson \rightarrow Univ \rightarrow Prop$.

The context type in the role type $Student$ (3) constrains the set of proof objects for $LegalPerson$ to a set of players for this particular role. Among all proof objects for the player of the role, only those for which both the category $Univ$ and the predicate $RegisteredIn$ are proved will be validated. As a consequence, the set of proof objects for the player of the role is a subset of the set of proofs for $LegalPerson$. This definition of a student role type complies with the presupposition (i.e., registered) provided in the role context. In the constructive setting, we first have to prove the relation. Let $x\ :\ Univ$ having the value US, where US is a proof object for the University of Savoie. The predicate $RegisteredIn$ represents the type of proof witnessing for the status "registered", i.e., a truth value whenever the person is a student. For instance, the list of proof objects for $Student$ will look like e.g., $\{\ldots \langle UF14553, \langle US, p_{12}\rangle\rangle, \langle UF14554, \langle US, p_{13}\rangle\rangle, \langle UF14557, \langle US, p_{14}\rangle\rangle, \ldots\}$ where p_{12} is a proof for $RegisteredIn\ (UF14553, US)$ and $UF14553$ represents a person through its registration number. According to that list, the first term of the nested pair supplies the instances of the player, e.g., $UF14553$,

$UF14554$, etc. If some time after having checked these values we check again the proof objects and observe that the proof object $UF14553$ does not occur, then we can conclude that the related person is no longer a student within this university (the player has left the role). Two aspects are worth noticing about the player of the role, (i) it has the type $LegalPerson$ in this context and (ii) its proof objects provide the set of players for that role (not only are we able to prove the role $Student$, but we can list all its corresponding instances). The Coq code for such a role definition requires the definition of a category (i.e., $LegalPerson$),

```
Definition LegalPerson          : SAG   := Type.
Definition Univ                 : SC    := Type.
Variable RegisteredIn           : forall (x:LegalPerson),(forall (y:Univ), Prop).
Definition C(x:LegalPerson)     := { u : Univ & (RegisteredIn x u)}.
Definition Student              := { x : LegalPerson & (C x)}.
Hypothesis tuple1               : Student.
Check Student.
Check (projT1 tuple1).
```

When checking the type of the role $Student$, the Coq answer is[22]:

```
Student           : Type (* max(role_student.9, role_student.8) *)
projT1 tuple1     : LegalPerson
```

It means that the resulting universe of the type $Student$ is assumed to be the maximum universe between those of $LegalPerson$ and $Univ$ as stated in section 2.5. Proofs for the role player which are extracted with the first projection (i.e., π_1) are of the type $LegalPerson$ as expected.

6.2. Discussion

With this definition of a role structure, several aspects are worth to be detailed. First, the concept of anti-rigidity for a role inherited from anti-rigid property types refers to the possible existence of a proof object for the player of the role (at a given time). As explained in subsection 2.3, each hypothetical judgment is the K-DTT translation of the modal possibility. For example, in an environment Γ assuming the existence of a Rol type, the judgment $\Gamma \vdash r : Rol$ introduces a potential role r without providing any proof for it. Second, the existence of a role instance requires simultaneously the existence of a player and a context detailing the conditions that the player has to fulfill in order to be able to play the role. In K-DTT, a role ceases to exist if the player and the context does not exist. This aspect relates to the ontological nature of the context which inherits this property from the encapsulating role, itself subsumed by the property type seen as a moment universal (see subsection 4.2). This view contrasts with the view of Baldoni-et-al (2006) where a role ceases to exist, even if both its player and the context which interacts with him exist. The major reason for this divergence is the dependence of a context on something like an activity since we cannot speak of a context in isolation but always in reference to something Dourish (2001). Third, in contrast with the Object-Oriented framework in which objects are built by the properties of a class, in K-DTT roles are dynamically checked by an algorithm of type inhabitation and are proved (or not) at a given iteration of the algorithm. The position defended in Baldoni-et-al (2006) is along this line with a context in which objects interact only via the roles played with the help of the affordances offered by their roles. Fourth, role players correspond to a restricted version of the type P, i.e., the subset of P's for which the role context C is valid. This definition also agrees with the view of Loebe (2007) where roles are existentially dependent on their players (each proof for the role is a pair which requires both a proof for the context and a proof for the player type). Fifth, with such a definition role types appear as a particular (a restricted) version of anti-rigid property types, an assumption which is in line with the approach of Baldoni-et-al (2007) but contrasting the view of Steimann (2007). This last property of roles provides a possible answer to the open question concerning the relation between roles and relationships Boella-et-al (2007).

[22] In the Coq syntax, `role_student.n` refers to the nth variable in the program called `role_student.v`.

While social roles capture some individual objects on a social ontological level, one may wonder whether the above definition also applies for other kinds of roles such as processual roles as they are introduced in Loebe (2007), i.e., processual roles are parts of processes. A particular property of processual roles is that they are two entities which are part of the process, i.e. the actor and the subject of the process. Therefore there can be two context types according to the perspective that is adopted, either from the actor or from the subject. As a consequence, two role types can be defined and related. Let us explain these aspects on the example given in Loebe (2007). In the process of John's moving a pen, there is an actor (John) and a subject (the pen). The player John belongs to the category $LegalPerson$ while its context involves both the subject and process from which the processual role is part-of and the predicate $Moves()$ as follows:

$$C[x : LegalPerson] \triangleq \Sigma s : Pen\ .\ Moves[x,\ s]$$
$$Mover \triangleq \Sigma x : LegalPerson\ .\ C[x]$$

with $Moves : LegalPerson \rightarrow Pen \rightarrow Prop$. Here, $Mover$ describes a process in which a person moves a pen. Alternatively, from the perspective of the pen, there is a context in which the pen is moved by John. A context type C' and its corresponding role type $Moved$ will capture these assumptions with:

$$C'[x : Pen] \triangleq \Sigma l : LegalPerson\ .\ Moves[l,\ x]$$
$$Moved \triangleq \Sigma x : Pen\ .\ C'[x]$$

These two role expressions where the player is either the actor or the subject can be seen as parts of the situation type: $S \triangleq \Sigma x : LegalPerson\,.\,\Sigma y : Pen\,.\,Moves[x, y]$. It is worth noticing that they also reflect the dual nature of processual roles. The author also introduces relational roles describing the way in which an argument participates in some relation. The K-DTT also complies with this role type with the player as argument and the context as the remaining part of the relation. Then, the view of roles as categorized in three categories (i.e., social, processual and relational) can be easily expressed in a single structure with the K-DTT theory which provides a uniform approach to the concept of role.

For some applications of roles such as RBAC (Role-Based Access Control) Sandhu-et-al (1996) and the OO approach Baldoni-et-al (2005), the definition of a role has to specify both what is required to play the role and which capabilities (also called affordances) the players will have when playing the role. We argue that such a position can be accounted for with role preconditions and role postconditions. The preconditions stand for what we have called so far "context type". By postconditions, we mean that a role can represent e.g., competency to do specific tasks or may embody authority and responsibility. These postconditions can be treated as properties of a given role type. Then we can separate mandatory properties and possible properties with the respective rigid and anti-rigid properties introduced in subsection 4.5. The structure is similar, however the major distinction holds in the fact that preconditions are put inside a role (context type) whereas postconditions depend on the role and are external to the role. It follows that objects may behave differently when participating in different roles. For example, being a student entails that (s)he may pass some examination. It could be represented as the following anti-rigid property:

$$PassExam \triangleq \Sigma x : Student\ .\ \Sigma y : Examination\ .\ Pass[\pi_1 x, y]$$

The properties of the role does not include the properties from its player since the relation between a role type and a player type is not a subsumptive relation but rather a subset relation (see e.g., subsection 7.3). The postconditions of a proof object playing a role are the postconditions which are valid for the role itself.

The role type fully captures the notion of founded universal Guarino (1992) since it depends on a value of another (external) category. Here, the role $Student$ depends on the object US having the category $Univ$ which is part of a relation supplied by the role context. Each of the identifiers in the list corresponds to a student, i.e., a player for the role student and is "founded" on the proof for the predicate $RegisteredIn$. All the required knowledge for the notion of role as described in subsection 5.2 and summarized by rule (2) is coherent with the syntactic structure of a role type through its salient properties:

- a role is expressed with a sum type,
- dependencies appear in the notion of parameterized context and

– players (of the role) are proof objects for the argument of the role.

Contrasting with some approaches, e.g., Sowa (1988), the player of the role is not a subtype of a category (natural type) but rather, we argue that the set of proofs for the player is a subset of the set of proofs of the natural type. Role types are declared statically but acquired dynamically. In other words, although the role types have to be declared during the design step (with their contexts), proof objects can populate roles at run-time depending on the content of the related database.

6.3. Properties of Role Players

In any role pair, as explained in subsection 6.1, the player of the role expresses a constrained version of the category described by P. This result can be easily generalized by induction on the set of proofs for any player type P.

Lemma 5. *(Proof inclusion) Given a set of proof objects for the role $r : Rol$ and a player category P such that $Rol \triangleq \Sigma x : P\,.\,C(x)$ where C denotes a role context type, then the set of proof objects for $\pi_1(r)$ is a subset of the set of proof objects for P.*

Proof. Let $\Theta_\Gamma(B)$ the set of all proof objects for the type B in Γ, that is, $\Theta_\Gamma(B) \triangleq \{b \mid \Gamma \vdash b : B\}$. Assuming a role (i.e., a proof object) $\langle p, c\rangle : Rol$ where $Rol \triangleq \Sigma x : P\,.\,C[x]$ and C describes a family of context types, let p_1 a given proof object and $\Theta_\Gamma(C[p_1/x])$, the corresponding set of proof objects. By summing on the n valid proofs $p_1, \ldots, p_n$ for x, the set of proof objects for the role R is the cardinal of the union $\Theta_\Gamma(C[p_1/x]) \cup \ldots \cup \Theta_\Gamma(C[p_n/x])$. Then using the π_1 elimination rule Luo (1994), we get a single proof for each value p_k, $1 \leqslant k \leqslant n$ since a given player can only appear at most once in each set of proofs for $\pi_1\langle p_k, c_k\rangle$. It follows that n is an upper bound for the cardinal of the set of proof objects for $\langle p, c\rangle$. Observing that the set of proof objects for the player $\pi_1\langle p, c\rangle$ is a constrained version of the type p, we get the final result $n \leqslant \mid \Theta_\Gamma(P) \mid$. □

If we consider the two roles for $Student$, e.g., $\langle JohnD, \langle US, p_{13}\rangle\rangle$ and $\langle JohnD, \langle UJF, p_{51}\rangle\rangle$ stating that $JohnD$ is both registered at the University of Savoie (US) and at the University of Grenoble (UJF), then it follows that (i) the two roles are distinct and (ii) the π_1 elimination rule extracts a single term, i.e., $JohnD$ proving that it is the common player for the two roles.

6.4. Role Types and Time

The notion of anti-rigidity captures the intrinsic potentiality in being an instance of a role and conveys (implicitly) a temporal aspect. The relation between timing information and the notion of anti-rigidity can be defined in the following way. A concept is anti-rigid if, assuming a discrete time t, a role type is proved for some values of t while it is not proved at other time instants. Role types can be annotated with temporal information for all applications that need time information. The type of t can be extracted from standards (e.g., the XML-supported date format ISO-8601) in which standard time zones and leap years are taken into account. Qualitative points, metric points and intervals may be either dates and times like "2005-02-10" or periodic like "day". The data type DateTime is intended to specify points while the data type TimeDuration is intended to specify intervals. Above these assumptions, it appears that type checking approaches including subtyping are well-suited to relate different time-based types Bry-et-al (2005).

In K-DTT, it is possible to consider time just as another variable. For example, endurants are in time and they have a temporal extension. The theory provides a set of type constructors to define time-based types like "day", "week", or "working day". Such types are declared by defining predicates. Dependent types easily put constraints on different parts of a date. Let us cite an example extracted from Jacobs (1999) for representing in a concise and precise way the Gregorian date.

$$GregorianDate \triangleq \Sigma y : N\,.\,\Sigma m : Nat[12]\,.\,Nat[length\ of\ month\ m\ in\ year\ y]$$

where $Nat[n]$ is the type of natural numbers ranging from 1 to n and $Nat[length\ of\ month\ m\ in\ year\ y]$ is defined by cases. In such a way a (Gregorian) date is a dependent tuple, e.g., $\langle 2011, \langle 8, 17\rangle\rangle$. Notice that,

due to the Curry-Howard isomorphism, correctness of a date representation becomes a well-typedness issue.

The role definition can be extended by adding a variable (e.g., t) having a time-based type to the list of arguments of the constraint. In the student example, the constraint $C(x)$ becomes:

$$C[x : LegalPerson] \triangleq \Sigma u : Univ\ .\ \Sigma t : TimeDuration\ .\ RegisteredIn[x,\ u,\ t]$$

with $RegisteredIn : LegalPerson \rightarrow Univ \rightarrow TimeDuration \rightarrow Prop$. On June 2011 we can extract the list of proof objects from the university database e.g., $\{\ldots \langle UF14553, \langle US, \langle 2010 - 2011, p_{12}\rangle\rangle\rangle, \langle UF14554, \langle US, \langle 2010 - 2011, p_{13}\rangle\rangle\rangle, \langle UF14557, \langle US, \langle 2010 - 2011, p_{14}\rangle\rangle\rangle, \ldots\}$ (since the time type for high-schools is covering two years). Now, if we consider the role type in September 2011, we get a new list of proof objects in which students have disappeared while others are added for the role type $Student$. It means that the disappeared players with corresponding identifiers are no longer proofs for the role $Student$. In other words, the role type (through its constraint) is indexed on time values, while its players (i.e., proof objects) can change accordingly.

The type inhabitation process extract proofs (data) from a database and values are submitted to the same constraints as for the temporal databases with explicit time models (see e.g., MacKenzie (1991)). The process of type inhabitation needs a minimum time interval to complete and processes whose execution time limits are above this time interval cannot be accounted for. This is a limitation of K-DTT but whose target framework impacts essentially real time based systems.

7. Subsumption with Roles

One major interest of role types relies on the possibility to extend them in order to cope with more subtle representations such as complex roles or the role of a role. There are in general two conceptual ways to extend knowledge about a structure, i.e., intensionally by adding properties or extensionally by adding data to the structure. These two notions will be respectively fulfilled by the subsumption and the partonomic relations[23] and we focus here on the intensional view, i.e., subsumption. The subsumption between categories has been introduced in section 4.7. The subsumption of role types and its properties requires first to discuss how relation types are handled with subsumption since role types involve context types themselves made of relation types.

7.1. Subsumption between Relation Types

For the subsumption of relation types, we consider nested sum-types in which any argument of the sum type can itself refer to a sum type. For any argument $x_i : T_i$, if the type T_i refers to a sum-type, then it can be nested up to a finite number of levels and we introduce $\vec{\gamma_i}$ with the same meaning as in subsection 4.5. Thus, an n-ary relation type would be expressed with the dependent sum: $\rho \triangleq \Sigma x_1 : T_1\ .\ \ \ldots\ .\ \Sigma x_n : T_n[x_1, ..., x_{n-1}]\ .\ P[\vec{\gamma_1}x_1, \ldots, \vec{\gamma_n}x_n]$, where the x_i are bound variables denoting categories T_i, $\vec{\gamma_n}$ stand for finite sequences of π_1 or π_2 and P is a predicate having the type $\tau_1 \rightarrow \tau_2 \rightarrow \ldots \rightarrow \tau_n \rightarrow Prop$ with $\vec{\gamma_1}x_1 : \tau_1$, ..., $\vec{\gamma_n}x_n : \tau_n$. In the case where x_i does not refer to a nested sum type, we have obviously $\tau_i = T_i$. From here on, we only consider relation types without nested sum-types because (i) simpler expressions enhance the readability, (ii) the introduction of nested argument types τ_i does not interfere with the lemma below and (iii) people interested in the nested version can refer to Dapoigny (2010b) where nested structures are detailed in context modeling.

Definition 10. *Given two relation types ρ and ρ' such that $\rho \triangleq \Sigma x_1 : T_1\ .\ \ \ldots\ .\ \Sigma x_n : T_n\ .\ P[x_1, \ldots, x_n]$ and $\rho' \triangleq \Sigma x_1 : T'_1\ .\ \ \ldots\ .\ \Sigma x_n : T'_n\ .\ P'[x_1, \ldots, x_n]$, then ρ subsumes ρ' if:*

[23] Also called part-whole relations.

(i) *for non dependent categories, all the properties of each type T_i also hold in T'_i,*
(ii) *for dependent categories, all the properties of each family type T_j also hold in T'_j, that is, the family T_j is a subfamily of T'_j,*
(iii) *the predicate P' is a sub-predicate of P (i.e., it is more precise).*

Lemma 6. *(Relation type subsumption) Given two relation types ρ and ρ' such that $\rho \triangleq \Sigma x_1 : T_1 \ . \ \ldots \ . \ \Sigma x_n : T_n \ . \ P[x_1, \ldots, x_n]$ and $\rho' \triangleq \Sigma x_1 : T'_1 \ . \ \ldots \ . \ \Sigma x_n : T'_n \ . \ P'[x_1, \ldots, x_n]$, if $T'_1 \preccurlyeq T_1$, $\ldots$, $T'_n \preccurlyeq T_n$, and $P' \preccurlyeq P$, then $\rho' \preccurlyeq \rho$.*

Proof. First, let us consider a binary relation type where we have two cases w.r.t. the dependence of arguments:

In the first case, the two arguments are non dependent. Then, rule 2 of the corollary 1 can be easily extended for a binary relation type $\Sigma x_1 : T_1 \ . \ \Sigma x_2 : T_2 \ . \ P[x_1, x_2]$, then the property holds for sum types with multiple arguments. Since relation types are expressed with sum types, the result also holds for relation types.

In the second case, we have $\Sigma x_1 : T_1 \ . \ \Sigma x_2 : T_2[x_1] \ . \ P[x_1, x_2]$. Since $T'_1 \preccurlyeq T_1$ and T'_2 is a subfamily of T_2 (i.e., $T'_2 \preccurlyeq T_2$), applying the rule 2 of corollary 1 yields that $\Sigma x_1 : T_1 \ . \ T_2[x_1] \preccurlyeq \Sigma x_1 : T'_1 \ . \ T'_2[x_1]$. Then applying again the corollary with $\Sigma \ x_1 : T_1 \ . \ \Sigma x_2 : T_2[x_1]$ and P standing respectively for A and B, one gets the final result.

Finally, the case of n-ary relation types is easily proved by induction over the structure of sum types. □

Let us consider, for example the relation $ParticipateIn$ described in Keet (2008). Individual qualities are elements of the DOLCE backbone so that one can refer to them directly in formal expressions. In the relation type, the predicate $ParticipateIn$ relates a first argument that must be an endurant (ED) together with a perdurant (PD). In K-DTT, a relation type $Participate$ requires the appropriate arguments within a sum type as follows:

$$Participate \triangleq \Sigma x : ED \ . \ \Sigma y : PD \ . \ ParticipateIn[x, \ y]$$

Now, let us consider the enzyme (endurant) that participates in a catalytic reaction (process). This is expressed as the relation type:

$$\Sigma x : Enzyme \ . \ \Sigma y : CatalyticReaction \ . \ ParticipateIn[x, \ y]$$

with $ParticipateIn : Enzyme \rightarrow CatalyticReaction \rightarrow Prop$. According to lemma 6, this relation type is also of type $\Sigma x : ED \ . \ \Sigma y : PD \ . \ ParticipateIn[x, \ y]$ since $Enzyme \preccurlyeq ED$, $CatalyticReaction \preccurlyeq PD$ and $ParticipateIn : ED \rightarrow PD \rightarrow Prop$.

7.2. Subsumption between Roles

The subsumption between role types looks very similar to the subsumption between relation types. This is not surprising, since in K-DTT context types that are part of role types are subsumed by relation types, as stated before (see subsection 4.2). Thus, we focus on the social and processual roles through a comprehensive example. For that purpose, let us revisit the example provided in Loebe (2007) where a medical unit forms a social context and creates its own patient role individuals. A patient in the clinical trial is treated within the medical unit by a physician. This example demonstrates the two aspects of the role structure (i) a social role (physician) where the player operates in the context of a medical unit and (ii) a processual role (patient) arranged with a clinical process defines the context initiated by the physician in which a person participates. The first role type is the social role involving a player whose category is $LegalPerson$:

$$\begin{array}{lcl} C1[x : LegalPerson] & \triangleq & \Sigma m : MedicalUnit \ . \ TryOutIn[x, \ m] \\ Physician & \triangleq & \Sigma x : LegalPerson \ . \ C1[x] \end{array}$$

with $LegalPerson \preccurlyeq SAG$, $MedicalUnit \preccurlyeq NAPO$ and $TryOutIn : LegalPerson \rightarrow MedicalUnit \rightarrow Prop$ (the DOLCE backbone is assumed to be already included in the basic context).

Then, the clinical process type is introduced as a process:

$$ClinicalProcess \triangleq \Sigma x : Physician\ .\ \Sigma y : ClinicalTrial\ .\ Perform[\pi_1 x,\ y]$$

from which the processual role type can be defined:

$$\begin{array}{lcl} C2[x : HumanPerson] & \triangleq & \Sigma p : ClinicalProcess\ .\ ParticipateIn[x,\ \pi_1\pi_2 p] \\ Patient & \triangleq & \Sigma x : HumanPerson\ .\ C2[x] \end{array}$$

with $ClinicalTrial \preccurlyeq Trial$, $Trial \preccurlyeq PRO$, $Perform : LegalPerson \rightarrow ClinicalTrial \rightarrow Prop$, $HumanPerson \preccurlyeq APO$ and $ParticipateIn : HumanPerson \rightarrow ClinicalTrial \rightarrow Prop$. The term patient is understood here as a processual role universal. This modeling decision is based on the fact that the patient is seen as a participant of a process (a clinical trial).

These definitions may be described at a higher level taking into account the previous subtyping relations.

$$\begin{array}{lcl} C'1[x : SAG] & \triangleq & \Sigma z : NAPO\ .\ ActIn[x,\ z] \\ SocialRole & \triangleq & \Sigma x : SAG\ .\ C'1[x] \\ TrialProcess & \triangleq & \Sigma x : SocialRole\ .\ \Sigma y : Trial\ .\ Make[\pi_1 x,\ y] \\ C'2[x : APO] & \triangleq & \Sigma p : TrialProcess\ .\ InvolvedIn[x,\ \pi_1\pi_2 p] \\ ProcessRole & \triangleq & \Sigma x : APO\ .\ C'2[x] \end{array}$$

with $ActIn : SAG \rightarrow NAPO \rightarrow Prop$, $Make : SAG \rightarrow PRO \rightarrow Prop$ and $InvolvedIn : APO \rightarrow Trial \rightarrow Prop$[24]. According to lemma 2 of subsection 3.1, the universe of each role type is computed as the maximum of its composing universes and provided that $TryOutIn$, $Perform$ and $ParticipateIn$ are the respective sub-predicates of $ActIn$, $Make$ and $InvolvedIn$, then lemma 6 can be applied with the results:

$$Patient \preccurlyeq ProcessRole \quad \text{and} \quad Physician \preccurlyeq SocialRole$$

The following fragment of Coq code proves all these aspects (contexts are implicit in order to simplify the writing).

```
Definition MedicalUnit    : NAPO            := Type.
Definition HumanPerson    : APO             := Type.
Definition LegalPerson    : SAG             := Type.
Definition Trial          : PRO             := Type.
Definition ClinicalTrial  : Trial           := Type.

Hypothesis TryOutIn       : forall (x:LegalPerson),(forall (y:MedicalUnit), Prop).
Definition Physician      : SAG := {x:LegalPerson & C1(x)}.
Hypothesis Perform        : forall (x:LegalPerson),(forall (y:ClinicalTrial), Prop).
Definition ClinicalProcess: Trial := {x:Physician & ({y:ClinicalTrial &
                                                      (Perform (projT1 x) y)})}.
Hypothesis ParticipateIn  : forall (x:HumanPerson),(forall (y:ClinicalTrial), Prop).
Definition Patient        : Trial := {x:HumanPerson & C2(x)}.
Hypothesis ActIn          : forall (x:SAG),(forall (y:NAPO), Prop).
Definition SocialRole     : SOB := {x: SAG & C'1(x)}.
Hypothesis Make           : forall (x:SAG),(forall (y:PRO), Prop).
Definition TrialProcess   : PRO := {x:SocialRole & ({y:Trial & (Make (projT1 x) y)})}.
Hypothesis InvolvedIn     : forall (x:APO),(forall (y:Trial), Prop).
Definition ProcessRole    : PRO := {x:APO & C'2(x)}.

Hypothesis test1 : POB->Prop.
Check (test1 ProcessRole).
Check (test1 Patient).
Hypothesis test2 : SOB->Prop.
Check (test2 SocialRole).
Check (test2 Physician).
```

[24]Notice that the predicate names may be more carefully selected on the basis of terminological analysis.

and whose results are:

```
test1 ProcessRole     : Prop
test1 Patient         : Prop
test2 SocialRole      : Prop
test2 Physician       : Prop
```

As it can be noticed, it proves that the subtyping rule (Sub) holds between the complex role types $Patient$ and $ProcessRole$ since the type ($Patient$) can be used instead of its parent type ($ProcessRole$). The same arguments hold for $Physician$ and $SocialRole$.

Corollary 2. *(Proof inference) Given two relation types ρ and ρ' such that $\rho' \preccurlyeq \rho$, then any proof that holds for ρ' also holds for ρ.*

Proof. Since $\rho' \preccurlyeq \rho$, it is obvious to see that any proof of type ρ' is also of type ρ up to the subtyping relation. □

Turning back to the role type $Physician$, assume that a database provides a proof object $MU1$ for $MedicalUnit$ and a list of proof objects for the term of type $LegalPerson$, e.g., $P0247, HT155, P0801, HP243, \ldots, HP122, \ldots$. In this list which collects all persons standing in $MU1$ (within a given time interval), persons having a $Pxxxx$ identifier are patients, persons having a $HPxxx$ identifier denote physicians operating in $MU1$ and persons corresponding to $HTxxx$ identifiers belong to the technical staff. Then, the constructor of the last term $TryOutIn$ will filter the previous list and provide a type $Prop$ that is only proved for $HPxxx$ identifiers. For example, some proof objects for the role $Physician$ could be $\langle HP122, \langle MU1, p_1\rangle\rangle$, $\langle HP243, \langle MU1, p_2\rangle\rangle$, ... where p_1, p_2 denote the respective proofs for the types $TryOutIn[HP122, MU1]$ and $TryOutIn[HP243, MU1]$.

From corollary (2), it follows that $\langle HP122, \langle MU1, p_1\rangle\rangle$, $\langle HP243, \langle MU1, p_2\rangle\rangle$, ... are also valid proofs for $\Sigma x : SAG\ .\ \Sigma y : NAPO\ .\ ActIn[x, y]$. In other words, it means that we have proofs saying that some SAG are working within some $NAPO$.

7.3. Addressing the Generalization Hierarchies Problem

In Steimann (2000), the author has pointed out that some difficulties arise in specifying admissible types for roles that can be filled by instances of disjoint types. Then, he argues that the solution to this problem lies in the separation of role and type hierarchies. However, as underlined in Guizzardi (2005), this solution would result in a significant (and counterintuitive) revision of the UML meta-model. A running example starts from the fact that attempts to relate customer, person and organization within a single hierarchy is problematic. To solve this problem, Guizzardi has suggested an "ontologically correct" solution with the introduction of what is called "role mixins", i.e., abstract classes with disjoint subclasses (only subclasses can have instances). Two roles are defined, i.e., the roles "Individual customer" and "Corporate customer" are respectively subsumed by the kinds "Person" and "Organization" whereas a role mixin "Customer" subsumes both "Individual customer" and "Corporate customer". This solution appears more interesting since it explicitly states that role mixins are relationally dependent (thus, giving the so-called context). Furthermore, it can be incorporated in a conceptual modeling design pattern without difficulties. What is lacking is the theoretical part making these conceptual choices effective and provable. For that purpose we suggest to express these choices in K-DTT.

Instead of an abstract class we introduce first a generic role $Customer$ involving the category ASO. Then, we have here two modeling choices for defining the predicate which will appear in each context type. On the one hand, it seems interesting to define a unique predicate at the highest abstraction level (i.e., $BuyTo : ASO \rightarrow ASO \rightarrow Prop$) and then to use polymorphism for all subsumed contexts. The main advantage of this solution holds in a significant code simplification. This is the solution adopted below for the theoretical description. On the other hand, in a more complex approach, each context uses its own predicate closely related to the type of its built in data. Its main benefit is a conceptually clearer model for the (conceptual) name which is used since each different predicate definition becomes independent

from data types, its name reflecting a different level. For example the highest level predicate is denoted $AskServiceTo$ and relates arguments of the category ASO instead of $BuyTo$ (with the same arguments) for the previous approach ($BuyTo$ should be conceptually restricted to a subset of ASO objects and lacks generality). This is the approach that has been explored in the code fragment given at the end of the subsection. The generic role type $Customer$ and its context is defined as:

$$\begin{aligned} C[x : ASO] &\triangleq \Sigma y : ASO\ .\ BuyTo[x,\ y] \\ Customer &\triangleq \Sigma x : ASO\ .\ C[x] \end{aligned}$$

Then, we introduce the categories $LegalPerson$ and $Organization$ which can be easily related to the DOLCE hierarchy observing that $LegalPerson$ is a Social Agent (SAG) and $Organization$ is a society (SC). Using the subtyping relations $Organization \preccurlyeq SC$, $LegalPerson \preccurlyeq SAG$, $SC \preccurlyeq ASO$ and $SAG \preccurlyeq ASO$ together with the transitivity of subtyping, it follows that $LegalPerson \preccurlyeq ASO$ and $Organization \preccurlyeq ASO$. The two roles "Individual customer" and "Corporate customer" are respectively denoted IC and CC and require the dedicated context types C_1 and C_2. These contexts share the common category $Corporation$ which the respective categories $LegalPerson$ and $Organization$ are customers of. Using corollary 1, the subtyping holds between these role types and their generic type $Customer$ w.r.t. the DOLCE hierarchy. The two roles types IC and CC subsumed by the generic role type $Customer$ are such that:

$$\begin{aligned} C_1[x : LegalPerson] &\triangleq \Sigma y : Corporation\ .\ BuyTo[x,\ y] \\ IC &\triangleq \Sigma x : LegalPerson\ .\ C_1[x] \\ C_2[x : Organization] &\triangleq \Sigma y : Corporation\ .\ BuyTo[x,\ y] \\ CC &\triangleq \Sigma x : Organization\ .\ C_2[x] \end{aligned}$$

With $BuyTo : ASO \rightarrow ASO \rightarrow Prop$, the IC role tells us that if one obtains proof objects for some corporation able to sell products e.g., Co_1 and some person e.g., $JohnDoe$, using polymorphism we get the proof q_1 for the predicate $BuyTo[JohnDoe, Co_1]$. The CC role is defined in a similar way. In other words, IC are constrained persons (customers of a given corporation) while CC are constrained organizations which are also customers of a given corporation. On the one hand, subsumption yields that $IC \preccurlyeq Customer$ and $CC \preccurlyeq Customer$ where $Customer$ is constrained by the common generic context type $C(ASO)$. On the other hand, the set of proof objects for the respective role player $\pi_1 x$ in $x : IC$ and $\pi_1 y$ in $y : CC$ are the respective subsets of the proof objects for the categories $LegalPerson$ and $Organization$ according to lemma 5. At the same time, the set of proof objects for the role player $\pi_1 z$ in $z : Customer$ is a subset of the category ASO. It follows that we have two parallel hierarchies, a role and a player hierarchies (see figure 3).

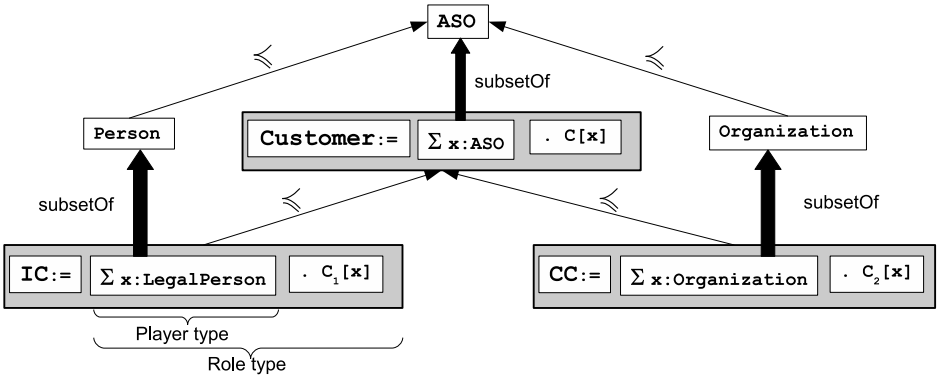

Fig. 3. Subsumption with roles in K-DTT.

With the same DOLCE categories, the role definitions are now described using the second approach with the following Coq code (contexts are not explicitly described for the sake of clarity):

```
Definition LegalPerson  : SAG   := Type.
Definition Corporation  : SC    := Type.
Definition Organization : SC    := Type.
Hypothesis AskServiceTo : forall (x:ASO),(forall (y:ASO), Prop).
Hypothesis IndivBuyTo   : forall (x:LegalPerson),(forall (y:Corporation), Prop).
Hypothesis CollecBuyTo  : forall (x:Organization),(forall (y:Corporation), Prop).
Definition Customer     : SOB := { x:ASO & { u:ASO & (AskServiceTo x u)}}.
Definition IC           : SAG := { x:LegalPerson & { u:Corporation & (IndivBuyTo x u)}}.
Definition CC           : SC := { x:Organization & { u:Corporation & (CollecBuyTo x u)}}.
Hypothesis Test         : SOB->Prop.
Check (Test Customer).
Check (Test IC).
Check (Test CC).
```

whose output is:

```
Test Customer           : Prop
Test IC                 : Prop
Test CC                 : Prop
```

This simple test proves that the subtypes IC and CC can be used instead of their parent type $Customer$. The dependent types make the difference with standard classes (types) of OO programming since we are able to treat simultaneously the two constraints without any extra programming technique or without separating hierarchies between classes as suggested in Steimann (2000). Furthermore, this solution avoids the introduction of mixins Guizzardi (2005); Loebe (2007) by using a more formal and elegant mechanism based on subtyping at the type-theoretical level.

8. Roles Properties

8.1. Identity Criteria for Roles

Another debate in role modeling is whether or not roles could be assigned an identity. We focus here on material roles, that is roles which have an identity. The issue whether roles carry their own identity has been addressed in many works and several authors Albano-et-al (1993); Kristensen (1995) claim that the identity of a role instance is inherited from a universal subsuming the role type whereas some others do not share this view (see e.g., Wieringa-et-al (1995); Steimann (2000)). Since the role has an IC, it follows that a role cannot subsume a property having no IC as pointed out in Welty (2001). In K-DTT, all proof objects can be distinguished and thus have an identity. Proof objects for the role type are distinguishable and their respective values count as an identification. It is clear that, for instance, as a student $Mike$ has a student ID and as soon as he ceases to be a student, then he loses that ID while always holding his own social insurance number (i.e., being a person). More precisely, one can get a proof object for the argument of type $LegalPerson$ in the example given in subsection 6.1 while there are no proofs for the predicate $RegisteredIn$ and as a consequence for the type $Student$. It follows that advocating dependent role types yields the acceptance of identity for players in line with the view of Guizzardi (2005). Notice that the player of a role has an identity only when playing the role.

8.2. Simultaneous Roles

Let us examine the case where an entity can play different roles simultaneously. For that purpose, one can introduce two different roles, a student and a worker with an additional constraint, i.e., a student in that case is also a worker. The second role depends on the first since we are interested in "simultaneous" roles, then, it is advised to first introduce a constraint $C(x)$ in which we restrict the student role (see subsection 6.1)

$$C[x : LegalPerson] \triangleq \Sigma u : Univ\ .\ RegisteredIn[x,\ u]$$

and then introduce the student role whose constraint forms the basis for the constraint of the worker role:

$$Student \triangleq\ \Sigma x : LegalPerson\ .\ C[x]$$
$$WorkStud \triangleq\ \Sigma x : LegalPerson\ .\ C'[x]$$

We have introduced the nested constraint type $C'[x : LegalPerson] \triangleq \Sigma y : C[x]\ .\ \Sigma z : Corporation\ .\ HiredAt[x, z]$ such that it requires the student context type $C[x]$. The result, i.e., the set of proof objects will enumerate all the students having a job concurrently to their studies.

Definition 11. *Two simultaneous roles are specified with first introducing a role type and then defining a second role type having (i) as player the same player as the first and (ii) as context type, a sum type depending on the first context w.r.t. the structure:*

$$Rol\ \triangleq\ \Sigma x : P\ .\ C[x]$$
$$SimRol\ \triangleq\ \Sigma x : P\ .\ C'[x]\ \ with\ \ C'[x : P] \triangleq \Sigma z : C[x]\ .\ \ldots\ .\ \phi[z, \ldots]$$

Major benefits of this definition are (i) the limitation of the search space by admitting only student objects in the new context, (ii) the flexibility allowing role types to be easily extended and (iii) the independence of subsumption of multiple role players from the number of roles.

8.3. Role of Role

If we consider the example given in Masolo-et-al (2004) where an employee plays the role of a project leader, the suggested solution solves the issue by considering that roles are in fact additive. Following that argument, roles of roles can be solved by specializing roles. This aspect can be related to the notion of complement for a role. In Loebe (2007), the author defines the notion of complement for a given role universal which says that a (universal) context is assigned to a set of role universals through the notion of "role base". In contrast with this relation-centered view, we rather consider a player-centered view where the player is the central part of the role. In this alternate strategy, a given context type C can be extended either by adding properties to C (the intensional perspective) or by extending its data structure with new information (the extensional perspective). Furthermore, different players may have different views on their environment and then, they may be related to different context types. This player-centered view is coherent with the assumption that context types describe moment universals.

More precisely, in K-DTT, we follow the view already explained in Dapoigny (2010b) which assumes (i) that a context type is related to a single intention (here, a role type) and (ii) that a context type is extensible either to the aggregation of disjoint context types or to a partonomic hierarchy of nested context types (see definition 4). It follows that each time a context is extended with new information then it is related to a distinct role type. Let us explain these assumptions on the example given in Loebe (2007), i.e., $John$ is a student of mathematics at the University of Leipzig. Using the $Student$ role of definition (3), a new role $StudentInField$ is introduced as a specialization of the $Student$ role:

$$C'[x : Student] \quad \triangleq \quad \Sigma c : CourseOfStudy[\pi_1\pi_2 x]\ .\ StudiesIn[\pi_1 x, c]$$
$$StudentInField \quad \triangleq \quad \Sigma x : Student\ .\ C'[x]$$

Notice that $CourseOfStudy$ depends on $\pi_1\pi_2 x$, that is a value which belongs to the category $Univ$. In other words we restrict the category $CourseOfStudy$ to be the course of study in the precise university where the student is registered. Then using the definition of $Student$, the π-elimination rules and variable renaming, we get the expression:

$$StudentInField \quad \triangleq \quad \Sigma l : LegalPerson\ .\ \Sigma z : C[l]\ .\ \Sigma c : CourseOfStudy[\pi_1 z]\ .\ StudiesIn[l, c]$$

The interesting result is that, with:

$$C"[l : LegalPerson] \triangleq \Sigma z : C[l]\ .\ \Sigma c : CourseOfStudy[\pi_1 z]\ .\ StudiesIn[l, c],$$

the resulting expression preserves the player and extends the context w.r.t. definition (3) from $C[l]$ to $C"[l]$. This result can be generalized from the equivalence between the two expressions: following the view of Masolo-et-al (2004), a role of a role seen as a specialization preserves the player and extends its initial context.

8.4. Revisiting the Counting Problem

The identity criteria have also a strong impact on the so-called counting problem, a recurrent problem involving role individuals with multiple instantiation Gupta (1980) which can be formulated in the following way:

- KLM served four thousand passengers in 2010.
- Every passenger is a person.
- Ergo, KLM served four thousand persons in 2010.

The assertions state that if a given person is registered in several flights of KLM in 2010, which is more than likely, the conclusion is false Wieringa-et-al (1995); Guizzardi (2005). This example has been throughly discussed, focusing on the relationships between the identity criteria (or coincidences) for passengers and persons. The basic problem is that counting passengers does not reduce to count persons, then a clear understanding of the relation person-role is required. Many solutions are given but what is needed is a comprehensive theory able to take into account all aspects of a role modeling.

Since they are passengers that can register more than once to several flights, counting persons cannot reduce to count passengers. This problem clearly refers to the issue whether roles carry their own identity and has been addressed in many works in which no consensus has emerged. However, it can be solved rigorously in K-DTT by using the following definitions:

$$\begin{array}{lcl} C[p : LegalPerson] & \triangleq & \Sigma t : Date\ .\ \Sigma f : Flight[t]\ .\ BookedAt[p, t, f] \\ Passenger & \triangleq & \Sigma p : LegalPerson\ .\ C[p] \end{array}$$

with $BookedAt : LegalPerson \rightarrow Flight \rightarrow Date \rightarrow Prop$. This role type describes a situation in which a reservation has been made on a given flight at a given date. For each proof of this type of situation there is a corresponding player $JohnD$, such that $\langle JohnD, \langle KL312, \langle Apr7 : 6PM, q_1 \rangle \rangle \rangle$ is a proof object for $Passenger$ where q_1 is a proof for $BookedAt[JohnD, KL312, Apr7 : 6PM]$. Using lemma 5, it is straightforward to deduce that the set of proof objects which occurs for the left argument of $Passenger$ is a subset of the set of proof objects for $LegalPerson$ and then, one can no longer say that KLM served four thousand passengers in 2010. Different proof objects for $Passenger$ correspond to different pairs but this does not exclude the first term (i.e., the player) to be identical. The conclusion, sharing the view of Guizzardi (2005), is that the principle of identity of passengers is supplied by a unique universal described by a free variable $x : LegalPerson$ such that the player described by the bound variable $x : LegalPerson$ within $Passenger$, is a subset of it.

8.5. Discussion

In order to summarize what can be exactly covered by the concept of role, we suggest to draw some distinctions between OO representations, FOL-based and the K-DTT view. For that purpose let us focus on some items that have been enumerated in subsection 5.2.

1. Many approaches consider roles are universals. For example, within an ontological perspective, roles are seen as specific classes Loebe (2007), in the OO domain they are bound to classes with relations Herrmann (2007) or collections of properties Kristensen (1995) while using FOL, roles are either manipulated through predicates acting on entities Masolo-et-al (2004) or considered as sorts in an order-sorted logic Steimann (2000). In Loebe (2007), the author has advocated for a categorization of roles into relational roles, processual roles and social roles. Assuming this view, we suggest to relate a role category with the category that its player belongs to. In such a way, the role definition (see

Definition 9) could be extended to express category-based roles subsumed by the general definition: $Role \triangleq \Sigma x : PT \,.\, C[x]$.

$$SocialRole \triangleq \Sigma x : ASO \,.\, C[x]$$

$$ProcessualRole \triangleq \Sigma x : ED \,.\, C[x]$$

We assume that players of processual roles are endurants.

2. Roles can be related to one or more behaviors. In K-DTT, behaviors can be introduced as role properties. However this possibility has not been investigated and is left for a future paper.
3. The problem of identity in a model supporting roles as distinguishable instances provides greater flexibility, because such a model admits dynamic structures that cannot be captured otherwise. We assume that roles have an identity different from their players since we define an identity for the role while maintaining the identity of the entity playing the role. Our approach is consistent both with the OO view, which clearly separates the role definition from the player's definition, and the FOL-based perspective using the so-called "qua-individuals"[25].
4. Roles are anti-rigid, that is role is not a permanent feature of an entity. A (realist) view would assume that a role exists iff there is (at least) a player together with an existing context. As underlined in Mizoguchi (2007), this assumption would lead to reject a model in which an instance of a role exists without being played by anything. The K-DTT theory solves this problem in the following way. A role type intensionally exists if we have a means to construct that type (e.g., having a structure relating this type with other types). This last aspect is in line with the view of Boella-et-al (2007) where the existence of a role type depends on the existence of a context type. A role type extensionally exists if we can collect proof objects for it (type inhabitation). Furthermore, the constructive view only argues for the existence of a means and not for the results of achieving this means, i.e., with proof objects.
5. As in most approaches, role types are relationally dependent, that is they require some external concepts to define them. The definition of roles types in K-DTT, embeds external concepts within the context of the role while the relation is arranged between a player and its context. By contrast with the OO approaches, in K-DTT the role is not an integral part of the object, it is a specific relation, or more generally, a specific interaction that can be proved for some existing objects. An entity can play a role concept only within a context. This property is a logical consequence of Definition (9) in which a context can range from a simple sortal to a complex set of relation types. The context type describes explicitly the requirements that a particular must fulfill to be a player for that role according to Husserl's notion of foundation Guarino (1992). Since a theory of context types has already been investigated (see Dapoigny (2010b) for more details), the definition of context types in K-DTT can take advantage of this study.
6. Different entities can play the same role at the same time. These entities are those accounted for in the list of proof objects corresponding to the player of the role and result from the existence of proof objects at a given time.
7. An entity can play multiple roles during a specific time interval. In K-DTT the proof objects for the player category are marked by the fact that they share a common role type (i.e., they are part of the tuple proving the role type) but nothing prevents some of these proof objects to occur in other role types provided that the category of the player is the same.
8. An entity can acquire and abandon roles (dynamic aspect). Proof objects can be involved in different role types, that is, in different interactions which are defined and then manipulated as types. The time is explicitly taken in account inside temporally characterized structures in which the temporal argument only constrains the validity of the structure with the usual ordering. Proof objects for all these types are quantified at each iteration of the type checking algorithm, with the intuitive consequence that K-DTT can hardly satisfy strong real-time constraints. However the dynamicity

[25] They refer here to the set of proof objects for the player of the role.

relies on the ability of proof objects to populate the different categories at each iteration of the type checker. In such a way any proof object is susceptible to shift from a typed structure to another one.

9. An entity can play the same role several times simultaneously. According to the interpretation given in Masolo-et-al (2004) a player is involved in several roles that are all specializations of a more general one. It can be considered as a particular case of the multiple roles issue which is easily represented in K-DTT.
10. A role can play another role. In the general definition (Definition 9), it is possible to define the player as a role type, but what is the expected meaning of such an assumption? As a consequence, we limit the semantics of K-DTT to additive roles as in Masolo-et-al (2004).

In OO programming languages, the status of role has been thoroughly investigated, e.g., in Bachman (1980); Steimann (2000); Herrmann (2007); Steimann (2007) where a great expressiveness for knowledge representation is highlighted. Among role-oriented approaches in software modeling, the programming language ObjectTeams/Java Herrmann (2007) is a significant one. While it technically supports the fact that roles come with their own behavior, this property appears more dedicated to OO programming and weakly addresses ontology modeling problems.

An alternative approach in OO programming considers roles as sets of affordances (see Baldoni-et-al (2006)). However, explaining roles in such a way, we have to collect all items of these sets for each role. A different perspective that is suggested here, consists in first expressing types of roles and then relating a minimal set of power (authority relationships) for each proved role. Types have a natural fit with observation since a type is inhabited if we collect in a database some proof objects having this type. Then, perceiving what objects afford boils down in K-DTT to populate the types which represent the affordances.

The first-order model of Masolo-et-al (2004) characterizes roles as the properties and relations reified at the object level for which it is possible to explicitly describe some aspects of the social contexts. Such a formalism could be easily translated in K-DTT with specifications but we do not follow this road here since we can support all the requirements for roles using higher-order capabilities. For example, the "Specialization" of roles is treated with subsumption while "requirements" generally correspond to the fact that roles can play roles (see subsection 8.3). Furthermore, unlike the first-order model, the K-DTT solution addressing the counting problem allows such a property to be derived from the modeling constructs used in representing knowledge (see lemma 5).

9. Conclusion

The obstacles standing in the way of the extension of a foundational ontology using expressive primitives are the lack of a unifying theory both sufficiently expressive and logically founded together with a logic which supports different abstraction levels. Due to its natural abstract character, it is a natural choice to describe an ontology in a high-level language. For that purpose, we have introduced the theory K-DTT which uses dependent types for their ability to support high expressiveness and powerful reasoning.

In a first part, we have shown how a type theory can be used to provide a highly-expressive language for ontologies. It is demonstrated that dependent types can model several non-trivial aspects of classes such as meta-level properties (rigidity, identity criterion, etc.). On the one hand, the present theory is more expressive than usual predicate logic in which it is neither possible to apply a function symbol to a proposition, nor to bind a variable except with a quantifier. In addition, the language of K-DTT is richer than the language of usual FOL-based systems in allowing proofs to appear as parts of the propositions so that the propositions can express properties of proofs (and not only of individuals like in FOL). This makes it possible to strengthen the axioms for existence, disjunction, absurdity and identity. On the other hand, the relation between OO programming and type theory relies on the ability of their representative computational structures to correctly express the semantics of ontological components. While their expressiveness is comparable, many aspects of object-oriented programming can be preserved in type theory since it unifies functional programming, component based programming, meta-programming (MDA), and logical verification (see Setzer (2007) for more details).

The second part aims at proposing a formalization of role with dependent types. The objective was to maintain an interdisciplinary character of roles by unifying ideas from prior research models within the logical framework and to give a more practical view of the abstract theory with (i) a significant number of examples and (ii) code fragments which detail the corresponding implementation with the Coq proof assistant. The role modeling is kept simple partly due to the fact that contexts are first class citizens of the theory (i.e., contexts as types). Notice that OO approaches provide solutions to role modeling that technically overlap K-DTT while lacking formal foundations and support for reasoning. This last aspect is very appealing for ontologies and is fully supported by K-DTT but it has not been developed here due to lack of place. However, the interested reader could find some examples on reasoning with dependent types e.g., for the correct specification of part-of relations in ontologies Dapoigny (2010a). The contribution includes a first formal solution for generalization hierarchies, identity criteria for roles, roles of roles and the counting problem.

We think that this paper can shed new light (i) on the ability of a type-theoretical language to support expressive ontologies and (ii) on a highly expressive contribution to role formalization. Some important problems have been left aside (e.g., role and behavior, reasoning about role types) due to the lack of space, but the K-DTT theory has the power to address them and it is planned for a future work.

References

Abadi, M. and Cardelli, L. (1998). A Theory of Objects, *Monographs in Computer Science*, Springer Verlag.

Akinkunmi, B. (2000). On the expressive limits of reified theories, *Journal of Logic and Computation*, *10*(2), 297–313.

Albano, A. and Bergamini, R. and Ghelli, G. and Orsini, R. (1993), An Object Data Model with Roles, *Proceedings of the 19th International Conference on Very Large Data Bases*, VLDB'93, 39–51.

Andrews, P. (1986). An Introduction to Mathematical Logic and Type Theory: to Truth through Proof, *Academic Press*.

Armstrong, J. (2009). In *Coders at Work: Reflections on the Craft of Programming*, 205–241, Peter Seibel, ed.

Artale A. and Franconi E. and Guarino N. and Pazzi L. (1996). Part-whole relations in object-centered systems: An overview. *Data & Knowledge Engineering*, *20*, 347–383.

Asher, N. (2008). A Type Driven Theory of Predication with Complex Types, *Fundamenta Informaticae*, *84*(2), 151–183.

Baader, F. and Calvanese, D. and McGuinness, D. and Nardi, D. and Patel-Schneider, P. (2003). *The Description Logic Handbook*, Cambridge University Press.

Bachman, C.W. (1980). The role data model approach to data structures, *Procs. of the International Conference on Data Bases*, S. M. Deen and P. Hammersley eds., 1–18.

Baldoni, M., Boella, G. and van der Torre, L.W.N. (2005). Social roles, from agents back to objects, *Workshop From Objects to Agents (WOA)*, 164–170.

Baldoni, M., Boella, G. and van der Torre, L. (2006). Modelling the Interaction between Objects: Roles as Affordances, *In Proc. of Knowledge Science, Engineering and Management, KSEM'06*, LNCS *4092*, 42–54, Springer.

Baldoni, M., Boella, G. and van der Torre, L. (2007). Relationships meet their roles in object oriented programming, *In Proc. of the 2nd International Symposium on Fundamentals of Software Engineering Theory and Practice (FSEN'07)*, LNCS *4767*, 440–448, Springer.

Barendregt, H. (1992). Handbook of Logic in Computer Science, *Lambda Calculi with Types*, *2*, Oxford University Press, 117–309.

Barendregt, H. (1997). The impact of the lambda calculus in logic and computer science, *Bulletin of Symbolic Logic*, *3*(2), 181–215.

Barendregt, H. and Geuvers, H. (2001). Handbook of Automated Reasoning, *Proof-Assistants Using Dependent Type Systems*, Elsevier and MIT Press, 1149–1238.

Barlatier, P. (2009). Conception et implantation d'un modèle de raisonnement sur les contextes basé sur une théorie des types et utilisant une ontologie de domaine, *Phd Thesis (in french)*, University of Savoie.

Bittner, T. and Donnelly, M. and Smith, B. (2004). Individuals, Universals, Collections: On the Foundational Relations of Ontology, *In Procs. of the International Conference on Formal Ontology in Information Systems (FOIS04)*, 37–48, IOS Press.

Boella, G. and van der Torre, L. and Verhagen, H. (2007). Roles, an interdisciplinary perspective, *Applied Ontology*, *2*(2), 81–88.

Boldini, P. (2000). Formalizing Context in Intuitionistic Type theory, *Fundamenta Informaticae*, *42*(2), 1–23.

Bruni, R. and Lluch Lafuente, A. and Montanari, U. and Tuosto, E. (2007). Service Oriented Architectural Design, *Procs. of the 3rd International Symposium on Trustworthy Global Computing*, LNCS *4912*, Springer, 186–203.

Bry, F. and Lorenz, B. and Spranger, S. (2005). Calendars and Topologies as Types, *Knowledge-Based Intelligent Information and Engineering Systems*, LNCS *3684*.

Calegari, D. and Luna C. and Szasz, N. and Tasistro, A. (2010). Representation of metamodels using inductive types in a Type-Theoretic Framework for MDE, *Instituto de Computación Facultad de Ingeniería Universidad de la República Montevideo, Uruguay*, tech. report RT 10-01.

Chen, P.P. (1976). The entity-relationship model: Towards a unified view of data, *ACM Trans. on Database Systems*, *1*(1), 9–36.

Cimiano, P. (2003). Translating Wh-Questions into F-Logic Queries, *Procs. of the 2nd CoLogNET-ElsNET Symposium*, 130–137.

Cirstea, H. and Coquery, E. and Drabent, W. and Fages, F. and Kirchner, C. and Maluszynski, J. and Wack, B. (2004). Types for Web Rule Languages: a preliminary study, *technical report A04-R-560*, PROTHEO - INRIA Lorraine - LORIA.
Cooper, R. (2005). Records and Record Types in Semantic Theory, *J. of Logic and Computation*, *15*(2), 99–112.
Coq Development Team (2010). The Coq Reference Manual, *Version 8.3.*, INRIA, France.
Coquand, T. and Huet, G. (1985). Constructions: A Higher Order Proof System for Mechanizing Mathematics, EUROCAL'85: Invited Lectures from the European Conference on Computer Algebra-Volume I, Springer-Verlag, 151–184.
Coquand, T. and Huet, G. (1988). The calculus of constructions, *Information and Computation*, *76*(2–3), 95–120.
Constable, R. L. (1991). Theoretical Aspects of Computer Software, *Type theory as a foundation for computer science*, LNCS *526*, Springer, 226–243.
Costa, P.D. and Guizzardi, G. and Almeida, J.P.A. and Pires, L.F. and van Sinderen, M. (2006). Situations in Conceptual Modeling of Context, *Procs. of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, 6–, IEEE Computer Society.
Dahchour, M. and Pirotte, A. and Zimányi, E. (2004). A role model and its metaclass implementation, *Inf. Syst.*, *29*(3), 235–270.
Dapoigny, R. and Barlatier, P. (2010a). Towards Ontological Correctness of Part-whole Relations with Dependent Types, *Procs. of the Sixth International Conference (FOIS 2010)*, 45–58.
Dapoigny, R. and Barlatier, P. (2010b). Modeling Contexts with Dependent Types, *Fundamenta Informaticae*, *104*(4), 293–327.
Dapoigny, R. and Barlatier, P. (2012). Formal Foundations for Situation Awareness based on Dependent Type Theory, *Information Fusion*, (accepted for publication), doi 10.1016/j.inffus.2012.02.006.
de Bruijn, N. (1987). Generalizing Automath by means of a lambda-typed lambda calculus, in D. Kueker, E. Lopez-Escobar and C. Smith (eds), Mathematical Logic and Theoretical Computer Science, Lecture Notes in Pure and Appl. Math., *106*, Marcel Dekker, New York, 71–92.
Dourish, P. (2001). Seeking a foundation for context-aware computing, *Human-Computer Interaction*, *16*(2-3), 229–241.
Enache, R. and Angelov, K. (2011). Typeful Ontologies with Direct Multilingual Verbalization, *Procs. of the Controlled Natural Languages Workshop (CNL 2010)*.
Gangemi, A. and Guarino, N. and Masolo, C. and Oltramari, A. and Schneider, L. (2002). Sweetening ontologies with DOLCE, *Procs. of the 13th International Conference Knowledge Engineering and Knowledge Management (EKAW2002)*, A. Gomez-Perez and V.R. Benjamins eds., LNCS *2473*, Springer, 166–181.
Galton, A. (1991). Reified Temporal Theories and How To Unreify Them, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, 1177–1182.
Girard J.Y. (1971). Une Extension de l'Interprétation de Gödel à l'Analyse, et son Application à l'Elimination des Coupures dans l'Analyse et la Théorie des Types Procs. of the Second Scandinavian Logic Symposium, Amsterdam, 63–92.
Girard J.Y. (1973). Quelques résultats sur les interprétations fonctionnelles, *Lecture Notes in Mathematics*, *337*, Springer.
Girard, J.-Y. (2006). The Blind Spot, *http://iml.univ-mrs.fr/ girard/coursang/coursang.html*.
Girard J.Y. and Lafont Y. and Taylor P. (1988). Proofs and Types, *Cambridge University Press*, *7*, Cambridge Tracts in Theoretical Computer Science.
Gödel, K. (1933). Eine Interpretation des intuitionistischen Aussagenkalküls, *Ergebnisse eines mathematischen Kolloquiums*, *4*, 39–40.
Guarino, N. (1992). Concepts, attributes and arbitrary relations, *Data & Knowledge Engineering*, *8*3, 249–261.
Guarino, N. (1994). The Ontological Level, In R. Casati, B. Smith and G. White (eds.), *Philosophy and the Cognitive Science*, Holder-Pivhler-Tempsky, 443–456.
Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation, *Int. J. Human-Computer Studies*, *43*(5–6), 625–640.
Guarino, N. (1998). Some Ontological Principles for Designing Upper Level Lexical Resources, *CoRR*, cmp-lg/9809002.
Guarino, N. and Welty, C. (2000). A Formal Ontology of Properties, Procs. of EKAW 2000, R. Dieng and O. Corby eds., LNAI 1937, Springer, 97–112.
Guarino, N. and Welty, C. (2002). Evaluating Ontological Decisions with OntoClean, *Communications of the ACM*, *45*(2), 61–65. ACM Press.
Guizzardi, G. and Herre, H. and Wagner, G. (2002). On the General Ontological Foundations of Conceptual Modeling, *Procs. of the 21th International Conference on Conceptual Modeling (ER2002)*, Springer-Verlag, 65–78.
Guizzardi, G. and Wagner, G. and Guarino, N. and Van Sinderen, M. (2004). An Ontologically well-Founded Profile for UML Conceptual Models, *In Procs. of the 16th Int. Conf. on Advanced Information Systems Engineering (CAiSE'04)*, Springer, 112–126.
Guizzardi, G. and Wagner, G. (2004). Towards Ontological Foundations for Agent Modelling Concepts Using the Unified Fundational Ontology (UFO), *AOIS*, 110-124.
Guizzardi, G. (2005). Ontological Foundations for Structural Conceptual Models, *University of Twente (Centre for Telematics and Information Technology)*.
Guizzardi, G. (2005). Agent Roles, Qua Individuals and the Counting Problem, *SELMAS*, 143–160.
Guizzardi, G. and Masolo, C. and Borgo, S. (2006). In Defense of a Trope-Based Ontology for Conceptual Modeling: An Example with the Foundations of Attributes, Weak Entities and Datatypes, *Procs. of ER'2006*, LNCS *4215*, 112–125.
Gupta, A. (1980). *The Logic of Common Nouns: an investigation in quantified modal logic*, PhD, Yale University.
Hancock, P. and Setzer, A. (2000). Interactive Programs in Dependent Type Theory, *CSL*, 317–331.
van Heijst, G. and Schreiber, A. Th. and Wielinga, B. J. (1997). Roles are not classes: a reply to Nicola Guarino, *Int. J. of Human-Computer Studies*, *46*(2), 311–318.
Herrmann, S. (2007). A precise model for contextual roles: The programming language ObjectTeams/Java. *Applied Ontology*, *2*(2), 181–207.
Heyting, A. (1930). Die formalen Regeln der intuitionistischen Logik, *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, 42–56.

Hickey, R. (2009). keynote at *JVM Languages Summit*.
Horrocks, I. and Patel-Schneider, F. (2004). A proposal for an OWL Rules Language, *Procs. of the Thirteenth International World Wide Web Conference (WWW 2004)*, 723–731.
Howard, W. A. (1980). To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, *The formulae-as-types notion of construction*, Academic Press, 479–490.
Jacobs, B. (1999). Categorical Logic and Type Theory, *Studies in Logic the Foundations of Mathematics*, *141*, Elsevier.
Keet, C.M. and Artale, A. (2008). Representing and reasoning over a taxonomy of part-whole relations, *Applied Ontology*, *3*(1–2), 91–110.
Kripke, S. (1963). Semantical Considerations on Modal Logic, *Acta Philosophica Fennica*, *16*, 83–94.
Kristensen, B. B. (1995). Object-Oriented Modeling with Roles, *Procs. of the 2nd International Conference on Object-Oriented Information Systems*, 57–71.
Krohs, U. (2011). Functions and fixed types: Biological and other functions in the post-adaptationist era, *Applied Ontology 6*(2), 125–139.
Loebe, F. (2007). Abstract vs. social roles - Towards a general theoretical account of roles, *Applied Ontology*, *2*(2), 127–158.
Luo, Z. (1990). A problem of adequacy: conservativity of calculus of constructions over higher-order logic, *Technical report ECS-LFCS-90-121)*, Department of Computer Science, University of Edinburgh.
Luo, Z. (1992). A Unifying Theory of Dependent Types: The Schematic Approach, *Procs. of Logical Foundations of Computer Science (LFCS'92)*, 293–304.
Luo, Z. (1994). Computation and Reasoning, *Oxford Science Publications*, *11*, International Series of Monographs on Computer Science.
Malecha, G. and Morrisett, G. and Shinnar, A. and Wisnesky, R. (2010). Toward a Verified Relational Database Management System, *37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 237–248.
Martin-Löf, P. (1982). Constructive Mathematics and Computer Programming, *Logic, Methodology and Philosophy of Sciences*, *6*, 153–175.
Masolo, C. and Borgo, S. and Gangemi, A. and Guarino, N. and Oltramari, A. (2003). Ontology Library (D18), *Laboratory for Applied Ontology-ISTC-CNR*.
Masolo, C. and Vieu, L. and Bottazzi, E. and Catenacci, C. and Ferrario, R. and Gangemi, A. and Guarino, N. (2004). Social roles and their descriptions, *In Proc. of the Conference on the Principles of Knowledge Representation and Reasoning*, AAAI Press, 267–277.
Masolo, C. and Guizzardi, G. and Vieu, L. and Bottazzi, E. and Ferrario, R. (2005). Relational Roles and Qua-individuals, *AAAI Fall Symposium on Roles, an Interdisciplinary Perspective: Ontologies, Programming Languages, and Multiagent Systems*, AAAI Press, 103–112.
McKenzie, E. and Snodgrass, R. (1991). An evaluation of relational algebras incorporating the time dimension in databases, *ACM Computing Surveys*, *23*, 501–543.
McKinna, J. (2006). Why dependent types matter, *Procs. of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, *41*(1), 1–1.
Mizoguchi, R. Ikeda, M. and Sinitsa, K. (1997). Roles of Shared Ontology in AI-ED Research – Intelligence, Conceptualization, Standardization, and Reusability, *Proc. of AIED-97*, 537–544.
Mizoguchi, R. (2007). *The model of roles within an ontology development tool: Hozo*, *Applied Ontology*, *2*(2), 159–179.
Montague, R. (1970). Pragmatics and intensional logic, *Synthèse*, *22*, 68–94.
Munn, K. Smith, B. (2008). Applied Ontology. An Introduction, *Metaphysical Research*, *9*, Ontos Verlag.
Muskens, R. A. (1996). Combining Montague Semantics and Discourse Representation, *Linguistic and Philosophy*, *19*, 143–186.
Napoli, A. (1992). Subsumption and classification-based reasoning in object-based representations, *Procs. of the 10th European Conference on Artificial Intelligence (ECAI92)*, 425–429, John Wiley & Sons Ltd.
Noble, J.R. (2010). Roles and Relationships, Procs. of the Dagstuhl Workshop on Relationships, Objects, Roles, and Queries in Modern Programming Languages.
Noonan, H. (2011). Identity, *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (ed.), <http://plato.stanford.edu/archives/win2011/entries/identity/>.
Van Paesschen, E. and De Meuter, W. and D'Hondt, M. (2005). Role modelling in SelfSync with warped hierarchies, *In Procs. of the AAAI Fall Symposium on Roles*, 149–155.
Poli, R. (2010). Philosophical Perspectives, Poli, R.; Seibt, J.; Healy, M.; Kameas, A. eds., *Theory and Applications of Ontology*, *1*, Springer.
Prawitz, D. (1965). Natural Deduction: A Proof-Theoretical Study, Almquist & Wiksell eds., Dover Publications.
Oury, N. and Swierstra, W. (2008). The power of Pi, *SIGPLAN Notices*, *43*(9), ACM, 39–50.
Ranta, A. (2004). Grammatical Framework: A Type-Theoretical Grammar Formalism, *Journal of Functional Programming*, *14*(2), 145–189.
Reus, B. and Streicher, T. (1993). Verifying Properties of Module Construction in Type Theory, *In Proc. MFCS'93*, LNCS **711**, Springer, 660–670.
Reynolds, J. (1983). Types, Abstraction and Parametric Polymorphism, *IFIP Congress*, Paris, 513–523.
Reynolds, J. (1984). Polymorphism is not set-theoretic. Semantics of data types, LNCS *173*, Springer, 145–156.
Rumbaugh, J. and Jacobson, I. and Booch, G. (1999). *The Unified Modeling Language Reference Manual*, Addison Wesley.
Russell, B. (1986) The Philosophy of Logical Atomism and Other Essays, 1914–19, John G. Slater ed., London: George Allen & Unwin.
Schürmann, C. (2001). Recursion for Higher-Order Encodings, Procs. of the 15th International Workshop on Computer Science Logic (CSL'01), 585–599.
Sandhu, R. and Coyne, E.J. and Feinstein, H.L. and Youman, C.E. (1996). Role-Based Access Control Models. *IEEE Computer*, *29*(2), 38–47, IEEE Press.

Searle, J. R. (1995). The Construction of Social Reality, *New York: Free Press*.
Setzer, Anton (2007). Object-Oriented Programming in Dependent Type Theory, *Trends in Functional Programming*, *7*, Intellect, 91–108.
Smith, B. and Rosse, C. (2004). The Role of Foundational Relations in the Alignment of Biomedical Ontologies, *MEDINFO 2004*, M. Fieschi et al. (Eds), Amsterdam, IOS Press.
Smith, B. and Ceusters, W. and Klagges, B. and Köhler, J. and Kumar, A. and Lomax, J. and Mungall, C. and Neuhaus, F. and Rector, A. L. and Rosse, C. (2005). Relations in biomedical ontologies, *Genome Biology*, *6*(5), R46.
Sowa, J. F. (1988). Using a lexicon of canonical graphs in a semantic interpreter, *Relational models of the lexicon*, Cambridge University Press, 113–137.
Sowa, J. F. (2000). Knowledge representation: logical, philosophical and computational foundations, *Brooks/Cole Publishing Co.*, Pacific Grove, CA, USA.
Steimann, F. (2000). On the representation of roles in object-oriented and conceptual modeling, *Data & Knowledge Engineering*, *35*(1), 83–106.
Steimann, F. (2007). The role data model revisited, *Applied Ontology*, *2*(2), 89–103.
Uschold, M. (1996). The use of the typed lambda calculus for guiding naive users in the representation and acquisition of part-whole knowledge, *Data & Knowledge Engineering*, *20*, 385–404.
Welty, C.A. and Guarino, N. (2001). Supporting ontological analysis of taxonomic relationships, *Data & Knowledge Engineering*, *39*(1), 51–74.
Werner, Benjamin (2008). On the strength of proof-irrelevant type theories, *Logical Methods in Computer Science*, *4*(3).
Whitehead, A.N. and Russell, B. (1997). Principia Mathematica, re-printed version.
Wieringa, R.J. and de Jonge, W. and Spruit, P.A. (1995). Using dynamic classes and role classes to model object migration, *Theory & Practice of Object Systems*, *1*(1), 61–83.
Woods, W.A. (1991). Understanding Subsumption and Taxonomy: a Framework for progress, in: *Principles of Semantic Networks*, J. Sowa Ed., Morgan Kaufmann, 45–94.